UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE ENGENHARIA GEOGRÁFICA, GEOFÍSICA E ENERGIA


# GeoToolsWeb: Uma Aplicação Web para Monitorização Estrutural

Carlos Miguel de Jesus Morais

**Mestrado em Engenharia Geográfica**

Dissertação orientada por:
Professor Doutor João Catalão Fernandes

2016

## Resumo

Este trabalho é uma amostra das diversas possibilidades de melhorar a cadeia de produção de monitorização estrutural. Neste caso, com a implementação de *software* que providencie soluções para uma pequena parte da monitorização estrutural.

O trabalho compreende duas fases, uma teórica e outra prática. A componente teórica descreve, como e que, métodos de monitorização estrutural serão utilizados, neste caso geodésicos. A componente prática passa pela implementação do *software*, neste caso uma aplicação web que seja moderna e apelativa em termos de visualização gráfica e funcionalidades.

Na componente teórica irá ser descrito como será tratada a informação e descritos os métodos que tratam a mesma. Na componente prática irá ser descrito que tecnologias, metodologias, grafismo visual e ferramentas foram usadas para atingir o propósito do primeiro parágrafo.

Por fim será elaborada uma conclusão do trabalho, sobre o que se aprendeu com todo este exercício e investigação do mesmo.



Palavras-chave:

**Monitorização Estrutural, Geodesia, Aplicações Web, Google Web Toolkit, Java**

## Abstract

This work aims to be one of many possibilities to improve the structural health monitoring production chain. In this particular case with the implementation of a web application in order to provide some solutions to the structural health monitoring area.

This work is separated in two phases, on theoretical and another practical. The theoretical describes, how and what, methods of structural health Monitoring will be used, in this case geodetic solutions. The practical phase is based on the planning and implementation of a web application that has to be appellative on graphics and functionalities, using the most modern technologies.

On the theoretical component will be described how will be processed the information and described what methods where used. On the practical component will be described what technologies, methodologies, graphism and tools where used to achieve the objectives of the first and second paragraphs.

At last, will be provided some tests to the application whit examples and will be elaborated a conclusion about what was learned with all the work and investigation.



Keywords:

**Structural Health Monitoring, Geodesy, Web Applications, Google Web Toolkit, Java**

# Agradecimentos

*Aos meus pais*

**Empresas onde trabalhei**

*Professor Doutor João Catalão Fernandes*

A minha memória tem esse momento marcado apesar de não ser perfeito, e por isso tudo o que sei de informática moderna começou algures aqui numa aula de cartografia dada nas práticas pelo Professor João Catalão Fernandes foi algo do género:

**JCF**: Olha, Carlos, não queres experimentar esta ferramenta da Microsoft que é diferente do que eles estão a fazer, de certeza que o Carlos percebe de programação apanha aquilo facilmente, faça o programa naquilo.

**CMM**: E perguntei o quê?

**JFC:** – "Abriu uma página web com o *download* um de visual studio aplicado à linguagem de para C#, da Microsoft, que era uma linguagem por objetos, depois então disse" – com isto, tem aqui a explicar, a documentação o *download*, tudo, faz nisto.

**CMM**: Eu vou ver melhor em casa. – "Fui ver melhor em casa, tive 19 no trabalho prático de cartografia, vinte? Não me lembro porque não, mas não foi de programação."

# Índice

## Índice de figuras

# Índice de esquemas

# Índice de tabelas

# Índice de exemplos

## Lista de abreviaturas

## **Informática**:

**GWT** – Google Web Toolkit

**Gm** – GeoMos

**Gmw** – GeoMosWeb

**GTweb** – GeoToolsWeb

**SGBD** – Sistema de Gestão de Base de Dados

**BC** – Business Component

**DAO** – Data Access Object

**HTML** – Hyper Text Markup Language

**PHP** – Hypertext Preprocessor

**CSS** – Cascading Style Sheets

**JS** – JavaScript

**MVP** – Model – View – Presenter

## **Engenharia Geográfica**:

**ME** – Monitorização Estrutural ou Estruturas

**Er** – Época de referência

**Eo** – Época de observação

**RP** – Rede Planimétrica

**MRP** – Micro Rede Planimétrica

**MQ** – Mínimos Quadrados

**A** – Matriz de coeficientes das observações nos mínimos quadrados

**W** – Matriz de fecho da função do ajustamento por mínimos quadrados

**L** – Vetor de observações nos mínimos quadrados

$\hat{\boldsymbol{x}}$ – Vetor de pontos a ajustar nos mínimos quadrados

# 1 Introdução

## 1.1 Objetivo do trabalho

O objetivo do trabalho é a implementação de uma aplicação web como uma solução de apoio à monitorização estrutural ou de estruturas (ME). A aplicação deverá ter a capacidade de ler, tratar e analisar dados geodésicos resultantes de operações de nivelamento geométrico e de medições de direção e distância de uma rede geodésica.

Esta aplicação tem que conter uma interface apelativa e moderna, recorrendo às mais modernas tecnologias web e metodologias apropriadas, que facilitem a sua melhor implementação, manutenção e compreensão.

## 1.2 Organização do trabalho

O trabalho está organizado por capítulos, começando por fazer uma introdução à monitorização estrutural, os instrumentos utilizados para o efeito e análise dos métodos geodésicos implementados. São analisados os algoritmos usados para o tratamento da informação dos métodos geodésicos aplicados.

É explicado o sistema aplicacional, como funciona e está arquitetado, de modo a cumprir os seus requisitos. São também analisadas as tecnologias usadas, o seu contexto e porquê.

Por fim são disponibilizados os resultados de testes aos algoritmos de ajustamento das soluções implementadas, já com o contexto aplicacional descrito para que fique percetível como a solução funciona e que a está correta.

## 1.3 Estado da arte

Das diversas pesquisas efetuadas nos jornais de referência e na web verificou-se que:

- Não existe muito *software* ME *online*, com exceção para o GeoMosWeb® (Gmw) da Leica®.
- Existem poucas grandes empresas de ME, como a Soldata®, Kinemetrics® e National Instruments®, que são as que se destacam com grande variedade de soluções, mas sem soluções web.
- Existe pouco *software* desktop para ME, disponibilizado aos utilizadores para compra, dado que as empresas da área disponibilizam apenas o serviço, mantendo o software internamente.

Desta forma pude constatar que o trabalho que iria realizar, apesar de pequeno e muito localizado em termos da área de ME, seria pioneiro na componente web, de modo a dar melhor funcionalidade e experiência aos seus utilizadores.

Verificou-se também que do ponto de vista teórico não há muito para investigar, mas sim para reaprender, querendo dizer com isto que não será acrescentado nada de novo nos métodos usados para tratar a informação geodésica. Do ponto de vista prático será planeada e implementada uma aplicação que que sirva os requisitos predefinidos, sendo esta sim pioneira e nova, pois será a única *online* com as mais modernas tecnologias web para a (ME).

# 2 Monitorização Estrutural

## 2.1 Introdução e conceitos importantes

A monitorização é um conceito muito abrangente, que envolve a observação / medição contínua de um fenómeno ou processo que tanto pode ser social, ambiental, biológico ou físico. Neste trabalho será abordado o processo de monitorização estrutural. A ME tem como finalidade fornecer, a qualquer instante do tempo de vida de uma estrutura, informação que permita a elaboração de um diagnóstico sobre o estado dos seus materiais, componentes e da estrutura como um todo. Consiste na execução de um conjunto de operações que observam as mesmas características da estrutura várias vezes ao longo do tempo, constituindo uma base de dados do estado de saúde da mesma.

A ME pode incidir sobre aspetos de uma estrutura, desde deslocamentos, provocados pela deformação de materiais, alteração do estado dos materiais, alterações estruturais, alterações no solo ou até acidentes. Na base da monitorização está a necessidade de segurança, precisão e rigor que o homem exige em seu redor, seja na construção ou manutenção de estruturas, que o leva a monitorizar diversos locais (2.1).

**FIGURA 2.1** – MONITORIZAÇÃO DE DIVERSAS ASPETOS DE ESTRUTURAS COM DIVERSOS EQUIPAMENTOS.

Os métodos e aparelhos usados para a monitorização estrutural são bastante variados e podem ser usados em simultâneo. Exemplo disso são níveis eletrónicos, estações totais, GNSS, acelerómetros, inclinómetros, piezômetros elétricos, fissurómetros, sensores de fibra ótica, sensores *wireless*, entre outros. A operação em simultâneo dos diferentes equipamentos com diferentes níveis de precisão possibilita a monitorização de diferentes aspetos da estrutura. Com base na interpretação dos diferentes resultados é possível identificar antecipadamente avarias estruturais ou situações de colapso que comprometeriam o tempo de vida útil da estrutura.

No contexto da geodesia, as diferentes técnicas têm como objetivo o posicionamento no espaço, de pontos materializados em determinada estrutura, relativos a um sistema de coordenadas de referência, geralmente local.

A precisão que se exige a estas coordenadas, e consequentemente às observações, é elevada, porque se procuram identificar pequenos deslocamentos com um elevado grau de confiança. As observações podem ter precisões na ordem das décimas de milímetro para distâncias e de décimas de segundo para direções, sendo estas precisões essencialmente obtidas por estações robóticas preparadas para o efeito.

Para tal, existem dois métodos que permitem realizar monitorização nestas condições, o nivelamento geométrico e a triangulação geodésica de micro redes geodésicas. Estes dois métodos vão ser abordados no trabalho, apesar de não serem os únicos implementados na solução aplicacional desenvolvida.

Com o objetivo de se conhecer a evolução do estado da estrutura ao longo do tempo, são realizadas com regularidade campanhas de observação. Estas constituem o conceito de época de observação ($\mathrm{Eo}$). Todos os projetos de monitorização contam com pelo menos uma época inicial, que é comum de ser chamada de época de referência ($\mathrm{Er}$) as restantes $\mathrm{Eo}$ realizadas ao longo do tempo com para comparação a $\mathrm{Er}$. Pode também adotar-se como época de referência a média entre épocas. No primeiro caso Er, coincide com a primeira época E0. Ou seja, observou-se a rede apenas uma vez e ficou estabelecida essa época como a época de referência.

Para o segundo caso a Er (2.1) é estabelecida por uma média percentual $\%\mathrm{n}$ entre as diversas épocas de observação feitas ao longo do tempo ($\mathrm{En}$), sendo a soma desta média um total de 100%. Cada época terá assim um contributo percentual n para a Er.

$$\mathrm{Er} = \sum_{0}^{n} \frac{\mathrm{En}}{\%\mathrm{n}} \tag{2.1}$$

A percentagem $\%\mathrm{n}$ é uma percentagem que agrava as precisões das observações de entrada para um novo ajustamento e nova ponderação das diversas épocas ($\mathrm{En}$). Isto irá alterar o critério de peso de ambos no ajustamento dos mesmos. Ou pode ser uma média ponderada dos pontos e suas precisões.

A determinação dos deslocamentos entre épocas, ou seja a variação do posicionamento dos pontos ao longo do tempo, permite determinar a evolução da deformação do material da estrutura a ser monitorizado.

A época de referência, permite comparar as suas coordenadas e erros com os das $n_s$ épocas que serão observadas no futuro, e desta forma temos o deslocamento dos pontos da época $n$, como mostra a equação (2.2).

$$\mathrm{Den} = \mathrm{P(Er)} - \mathrm{P(t)} \tag{2.2}$$

Em que P(t) é a posição 3D na época t. Uma época de referência (Er) é assim um conceito muito importante porque que irá ser usado em muitas situações e será a ligação a outros conceitos essenciais para o objetivo definido neste trabalho, a aplicação web. Desta forma é necessário conhecer todas as etapas para realização de uma época de observação, que são essencialmente o planeamento, levantamento de observações, tratamento da informação e disponibilização de um relatório final, de modo a saber corresponder essa informação de forma clara e simples para realização do projeto.

Como é normal num projeto, este divide-se em diversas fases. Durante algumas dessas fases a presença do Engenheiro Geógrafo é indispensável, pois ele é dos poucos com o conhecimento e técnicas de geodesia incluindo algumas soluções mais avançadas de GNSS.

Determinado o que se pretende monitorizar e sabendo os métodos que serão usados, geodésicos e se necessário geotécnicos, passa-se para uma outra fase em que a estrutura é analisada para que possa ser elaborado um plano de instalação de material de auxílio à monitorização, e a realização desta. No contexto da geodesia, esta fase compreende o estudo da rede de nivelamento ou uma micro rede de monitorização que irá ser implementada, onde são determinados o número de pontos a observar, o número de referências a usar e o número de estações a executar. Todas estas podem ser ou não referências, determinando assim a forma e dimensão da rede. Esta fase de planeamento é importante porque permite fornecer soluções de auxílio ao projeto, ainda que este seja executado antes de se estabelecer qualquer época de referência.

A época que corresponde às observações começa com o planeamento da campanha e observação efetiva da rede. Após esta fase é necessário tratar as observações obtidas da seguinte forma:

- Processamento e ajustamento da rede de observações;
- Análise e identificação de observações que possam estar a contaminar negativamente os resultados;
- Remoção de observações erradas (de menor precisão);
- Recomeço do processo se forem removidas observações;
- Comparação com a época de referência;
- Elaboração de relatórios e disponibilização de informação sobre diversas formas.

Torna-se mais evidente que o conceito de Er é um dos principais conceitos porque é em redor da Er que será comparada parte significativa da informação, desde observações, resultados, relatórios entre outras informações.

A frequência com que são realizadas épocas de observação é importante e tem impacto na aplicação, na medida em que as estruturas mais instáveis tendem a ser observadas com maior frequência em contraste com estruturas mais estáveis. Esta particularidade aumenta a quantidade de informação que é armazenada, com vista a ser consultada ou disponibilizada em qualquer momento.

É fundamental que a informação seja rapidamente armazenada, processada, e disponibilizada para consulta, em forma de gráficos e relatórios detalhados. Também é importante manter um sistema de alertas para erros, situações que necessitem da intervenção do utilizador ou apenas de notificações automáticas para disponibilização de nova informação. Em projetos de grande dimensão é recorrente utilizar monitorização em tempo real, que será abordada mais à frente, agravando assim a situação descrita em cima.

## 2.2 Processo de produção atual

As empresas que operam na área da monitorização tendem a ter projetos em diversos pontos do mundo que potenciam a utilização da internet como ferramenta de trabalho diária. Isto tornou possível uma maior rapidez em todo o processo de tratamento e disponibilização de informação, principalmente em mercados que sejam muito competitivos.

Ferramentas conhecidas, como Microsoft Office®, Microsoft Excel®, Matlab® e as que os fabricantes disponibilizam com os equipamentos, são as principais ferramentas de trabalho diárias. Efetuando uma busca na Google em inglês ou português, não é possível encontrar um sítio da web que se dedique pelo menos ao armazenamento e análise de dados de ME com exceção para o (Gmw).

Existem pequenos aplicativos que disponibilizam pequenas ferramentas web que envolvem ajustamentos de poligonais e até de redes geodésicas mas limitado no número de observações do sistema e difícil de usar, sendo apenas estas as suas funcionalidades, não dispensando outro *software* para analisar e comparar os resultados finais.

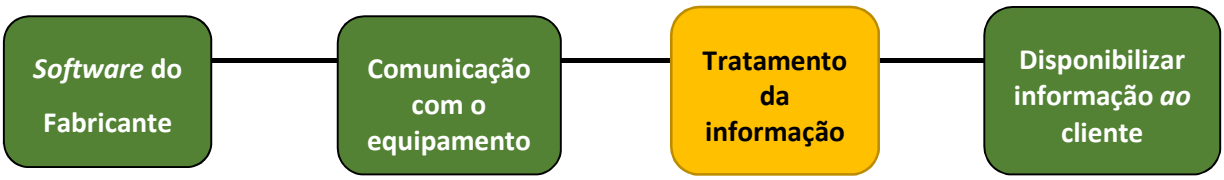
A conjugação dos factos acima tornou clara a necessidade de desenvolver uma aplicação web para ME , o GeoToolsWeb (GTweb). O GTweb permite, além da aplicação no *browser*, que sejam utilizados os seus *Webservices*, isto significa que estará em contacto com muitos operadores para além dos que usam o *browser* diretamente.

Um operador efetua uma observação de uma direção e de uma distância de um ponto, por exemplo, na Austrália e o outro observador tem acesso quase instantâneo, por exemplo, em Portugal, diretamente através do GTweb, ou de uma aplicação móvel usando os *Webservices* para mesmo o efeito, terá os dados analisados e toda a informação que necessita.

De facto, é possível que se possa fazer quase de tudo um pouco, por exemplo, mesmo sem recorrer diretamente ao *browser*, basta ter um programa que use o GTweb para que as etapas da produção da monitorização possam ter uma evolução diferente. Com o acesso à informação é possível programar tomadas de decisão imprevisíveis que impliquem voltar atrás na cadeia de produção e que precisam, neste caso por exemplo,

de novo ajustamento de observações, consulta de informação ou relatórios que necessite de envio ao cliente.

Equipamentos mais recentes, como algumas estações totais, incluem já a possibilidade de comunicar com os outros aparelhos via *bluetooth* e *wi-fi*, isto é, principalmente para melhorar a ME em tempo real. Uma aplicação móvel pode também controlar as mesmas funcionalidades da estação total, disponibilizar os dados via internet para o servidor, que seguirão o processo já abordado e por fim disponibilizados ao utilizador. Existem sistemas parecidos, mas que não recorrem a uma aplicação móvel para recolha e envio da informação, sendo geridos apenas por uma aplicação num $PC$, como servidor, pois outras ferramentas e recursos básicos necessários para uma solução melhor e mais sofisticada são relativamente caros de se manter pelas pequenas e médias empresas (PME's), que têm uma cadeia de trabalho, geralmente, simples como mostrado no esquema (2.1). As aplicações GeoMos® (Gm) da Leica® e o GTweb são duas tecnologias muito diferentes, ainda assim a união do GTweb com as ferramentas e funcionalidades do Gm, daria uma contribuição importante para a área, uma aplicação geodésica completa na Web.

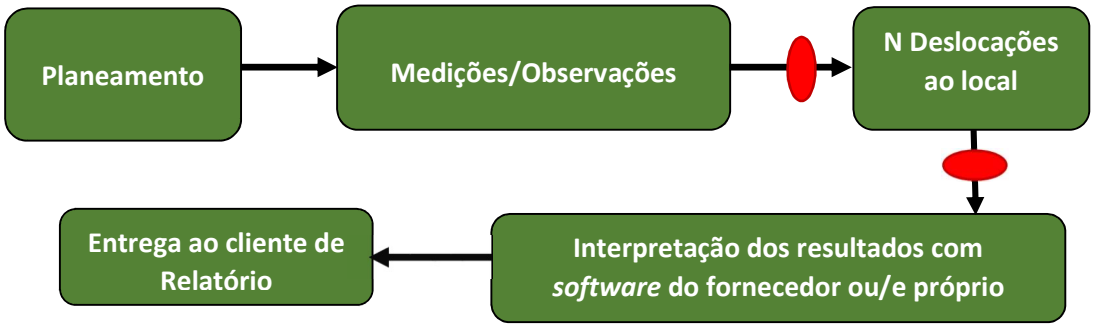



**ESQUEMA 2.1** – PROCESSO DE UM PROJETO DE MONITORIZAÇÃO DE ESTRUTURAS COMUM.

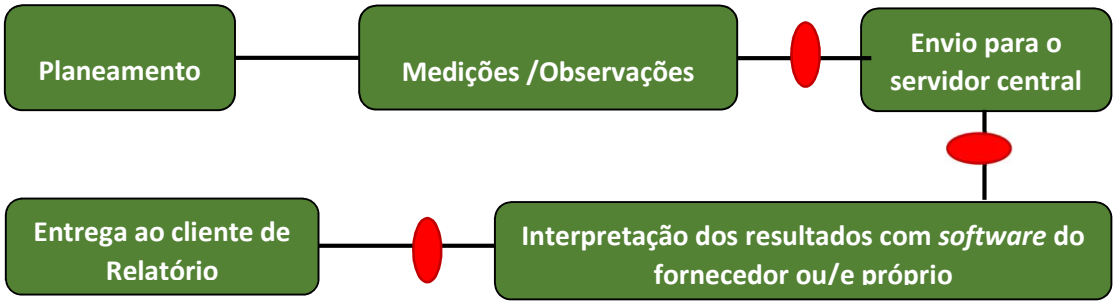
É importante notar que os diversos aparelhos de monitorização já referidos, têm *software* próprio do fabricante para tratamento da informação. Este facto tem impacto no processo, dado que tem que se replicar esse tratamento também, bem como na implementação de processos automáticos diferentes. No entanto, é mais fácil comunicar diretamente com os aparelhos e tratar a informação numa só aplicação.

## 2.3 Instrumentação e sistemas de monitorização estrutural

A cadeia de produção tradicional para monitorização estrutural ilustrada no esquema (2.2), e a cadeia de produção baseada na web no esquema (2.3), mostram o impacto positivo que a ME sofre derivado do uso da comunicação por internet, podendo assim serem incluídas outras etapas importantes como robotização e excluídas outras como as deslocações frequentes ao local, partindo do princípio que os instrumentos de observação estão sempre presentes.




**ESQUEMA 2.2** – PROCESSO DE UM PROJETO DE ME COM ETAPAS CÍCLICAS A VERMELHO.




**ESQUEMA 2.3**– PROCESSO DE UM PROJETO DE ME CENTRALIZADO COM ETAPAS CÍCLICAS A VERMELHO.

Sabendo as excelentes precisões associadas aos diversos equipamentos de monitorização e à quantidade de possibilidades que os mesmos permitem, duas questões se colocam:

a) Se existem instrumentos robóticos, sistemas autónomos e sofisticados, porque não existe uma aplicação web que que explore as possibilidades dos sistemas e equipamentos? Alguns aproximam ou dão apenas algumas funcionalidades como o Gmw, uma pequena parte do Gm, ou o Cyclops® da Soldata®, no entanto este último é *software* desktop e não licenciado ao público.

b) Se aqueles cujas tecnologias que utilizam para sistemas *online* ou aplicações desktop são antiquados, limitados ou difíceis de manter, tornando-os a longo prazo obsoletos. Porque não procuraram outras soluções? Será por falta de recursos que que não permitem inovação, compensação do projeto, concorrência na área ou outro fim?

Se interessado na resposta à questão seria lógico desenvolver um *software* com capacidade, não do equipamento mas, de toda a cadeia de produção, onde algures no seu processo entra o equipamento. A segunda ainda parece ser mais incomodante, porque na informática é costume dizer entre colegas e amigos "em informática tudo é possível se não violar as leis da matemática", e a verdade é que é, pode ser moroso, ter que se investigar, desenvolver e testar mas depois as finalidades e as possibilidades são infinitas.

Assim, pode ser concluído que o processo de ME tem poucas fases durante o seu projeto, que são relativamente simples de executar, o que é um ponto a favor, mas são muito morosas. Podemos também concluir que a realização de observações pode quase ser toda efetuada sem intervenção humana, em que os dados seriam tratados por a aplicação central.

Este processo pode ser alterado facilmente, para que as observações e outras informações debitadas pelo aparelho estejam no nosso completo controlo sem ter que depender de qualquer *software* externo, que geralmente vem com o respetivo equipamento.

Na verdade consoante maior for o grau de complexidade da rede, mais ambicioso se torna a vontade e possibilidade de utilizar e explorar novos métodos. Por exemplo, na imagem da esquerda da figura 2.2 temos uma rede complexa com apenas duas referências e quatro pontos a determinar, já na imagem da direita, temos uma poligonal, com um ponto a determinar e quatro referências. A complexidade da rede pode ter influência no processo, já que são feitas muitas observações, aumentando a probabilidade de erro humano.

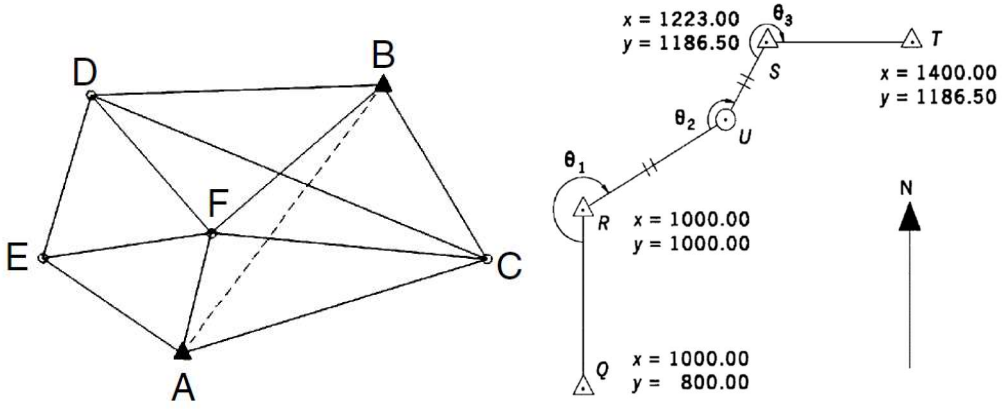



**FIGURA 2.2** – REDE COMPLEXA E FECHADA À ESQUERDA E REDE SIMPLES E ABERTA À DIREITA.

### 2.3.1 Instrumentação dos sistemas

Para a execução da monitorização são utilizados equipamentos como estações totais, GPS, inclinómetros, níveis eletrónicos, piezômetros elétricos, acelerómetros, sensores de fibra ótica, sensores *wireless*, fissurómetros, entre outros. Neste trabalho são descritos os equipamentos geodésicos que contribuem com medidas para as técnicas de nivelamento, triangulação e trilateração e inclinómetros, deixando de fora alguns como os acelerómetros, sensores de fibra ótica e sensores *wireless*.

**a) Inclinómetro**

O modo de operar os inclinómetros é mais simples, como também o tratamento dos seus dados e disponibilização dos mesmos. O aparelho efetua quatro medidas corrigidas, que são A, A+, B, B+, ou X, X+, Y, Y+, que são os valores das direções perpendiculares centradas num eixo orientado a uma direção N, como mostra a figura 2.3.

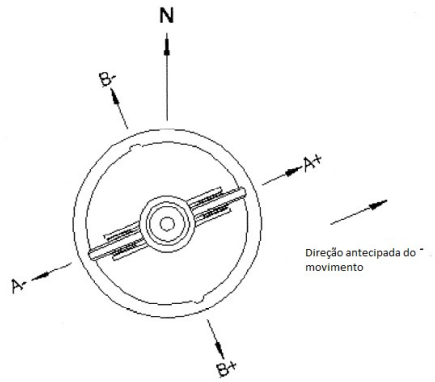



**FIGURA 2.3** – ESQUEMA DOS EIXOS DE UM INCLINÓMETRO.

**b) Nível e régua para medição.**

**FIGURA 2.4** – NÍVEL ELETRÓNICO E RÉGUA DE MEDIÇÃO.

O nível geométrico é um instrumento que mede distâncias e alturas em réguas próprias para o efeito (2.4), com o objetivo de calcular a diferença de altitude entre dois pontos (Desnível). Para tal a régua é uma régua especial e o equipamento, o nível, tem um sensor que mede a informação na régua, a altura da mesma, e o tempo que demorou a ir e a vir o sinal, já que o mesmo se aproxima à velocidade da luz $c$, obtendo assim a distância.

**c) Estações Totais**

As estações totais são assim chamadas porque medem direções e distâncias. São os aparelhos mais utilizados na ME e quase todos os sistemas autónomos as utilizam. Tal como todos os outros fabricantes, os de estações totais não ficam atrás nas elevadas possibilidades e precisões, construindo uma variedade grande de equipamentos bastando visitar o sítio na internet da Leica® pode-se constatar este facto.

Dado que o objetivo é monitorizar estruturas, então o equipamento tem que ser próprio para o efeito. Tem que se exigir uma precisão na distância bastante elevada, uma precisão na direção também elevada e na zenital o mais elevado que for possível. As melhores estações totais existentes no mercado têm precisão para uma direção azimutal de 0.5", uma zenital de 0.5" a 1.3" e para uma distância 0.6mm +1ppm. Existem outros fatores importantes nas estações totais como o nível de automatização ou o controlo remoto, bem como o tipo de prisma a colocar na estrutura, pois existem diferenças substanciais entre os mesmos.

A desvantagem de todos os fornecedores de estações totais é terem a comunicação com o aparelho bloqueada, de tal forma que se o utilizador quiser tornar a estação controlável a partir de *software* próprio terá que pagar para tal, ao contrário de restantes aparelhos, como é o caso dos fornecedores de inclinómetros, que disponibilizam todo o software necessário e a forma de comunicar com o aparelho. É claro que é possível recorrer a soluções e deixar de estar dependente dos fabricantes, sabendo todo o processo que o equipamento contem e o que o mesmo mede para se chegar aos resultados finais.

### 2.3.2 Sistemas avançados e *software*

Em ME é quase sempre empregue métodos clássicos, sem autonomia mas que por vezes é necessário, ou quase obrigatório, recorrer a métodos autónomos que observam permanentemente os alvos da rede. Estes métodos autónomos dependem de *software* que opere toda a cadeia o sistema.

Para os sistemas mais simples, onde são empregues os métodos clássicos, existem muitos programas, apesar do mais usual ser o Gm da Leica®, devido das suas possibilidades e popularidade que leva a uma elevada venda do mesmo tornando-os líderes de mercado.

No que respeita a sistemas que dispõem de ferramentas em tempo real, os mesmos contêm a possibilidade de controlar aparelhos que executem observações, armazenem e processem toda a informação em tempo real, como o que é produzido pelo Gm. O Gmw permite disponibilizar para a internet algumas funcionalidades e resultados do Gm.

São sobre os últimos sistemas, como o Gmw, que o trabalho se debruça. Um sistema que permita o armazenamento, tratamento, e disponibilização de informação de ME para a internet. Este sistema implicou um planeamento, implementação, verificação e testes, utilização e a disponibilização de uma aplicação web, o Gtweb.

## 2.4 Métodos geodésicos para monitorização estrutural

Nesta secção vão ser debatidos a análise e o tratamento do nivelamento geométrico e micro redes geodésicas.

### 2.4.1 Nivelamento geométrico

O nivelamento geométrico é um método, de medição direta, de alta precisão para determinação de desníveis e consequentemente de cotas. A precisão do método está dependente do rigor do aparelho, da graduação da escala das miras de observação, da distância entre os pontos, de eventuais erros de pontaria e a estabilidade da régua pelo portador da mesma, como mostra a figura (2.5). O facto de ser um método prático e simples de utilizar faz com que este seja empregue em várias áreas da engenharia, destacando-o aqui para a ME.

Os projetos de apoio à construção civil e monitorização de estruturas requerem nivelamento de alta precisão. A dimensão e complexidade do projeto determinam o número e comprimento de linhas de nivelamento a serem observadas, bem como o seu número de pontos. Para o caso de projetos de monitorização é de acrescentar ainda o facto de a mesma linha ser observada várias vezes ao longo do tempo.

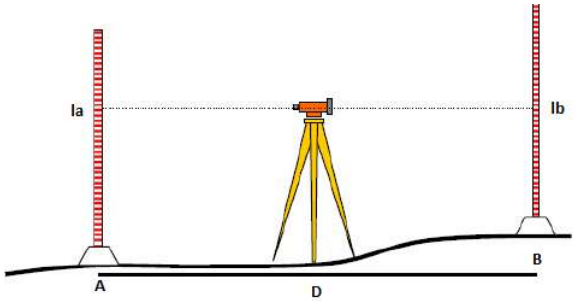



FIGURA 2.5 – OBSERVAÇÃO DE UM TROÇO DE UMA LINHA DE NIVELAMENTO GEOMÉTRICO.

Sabendo a cota do ponto A e com a da altura de B, pelas leituras nas réguas $la$ e $lb$, é determinado o desnível ente AB (2.3).

$$\Delta H_{AB} = l_a - l_b \tag{2.3}$$

Desta forma é possível o transporte de cota entre todos os pontos da linha de nivelamento (2.6), pela equação (2.4).

$$H_B = H_A + \Delta H_{AB} \tag{2.4}$$

Onde $H_{\boldsymbol{B}}$ é o ponto de cota a ser determinada e $H_{\mathrm{A}}$ é o ponto anterior de cota conhecida e $\Delta H_{A,\mathrm{B}}$ é o desnível calculado.

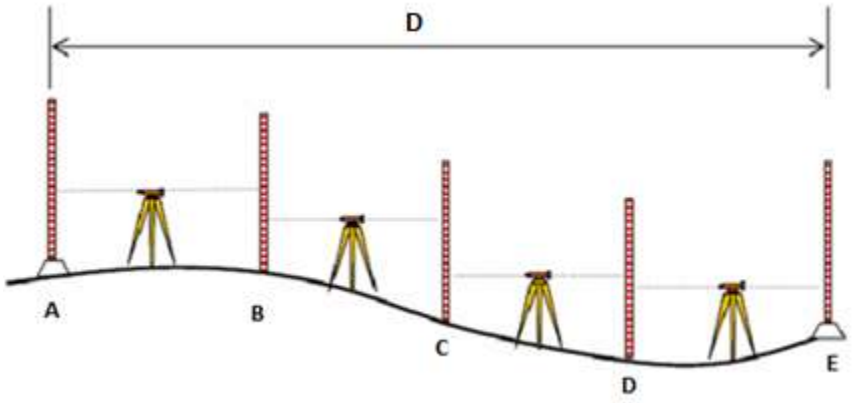



**FIGURA 2.6 – OBSERVAÇÃO DE VÁRIOS TROÇOS PARA TRANSPORTE DE COTA.**

O nivelamento pode ainda ser aberto ou fechado, ou seja, ou acaba e começa em referências diferentes, ou acaba na mesma referência, o Gtweb trata ambos os casos.

## 2.4.2 Micro redes planimétricas

As redes planimétricas (RP) (2.7) são redes triangulares planas, ou seja, são redes que dispõem de duas coordenadas, geralmente M (Meridiano) e P (Perpendicular) bem definidas e ainda distâncias.

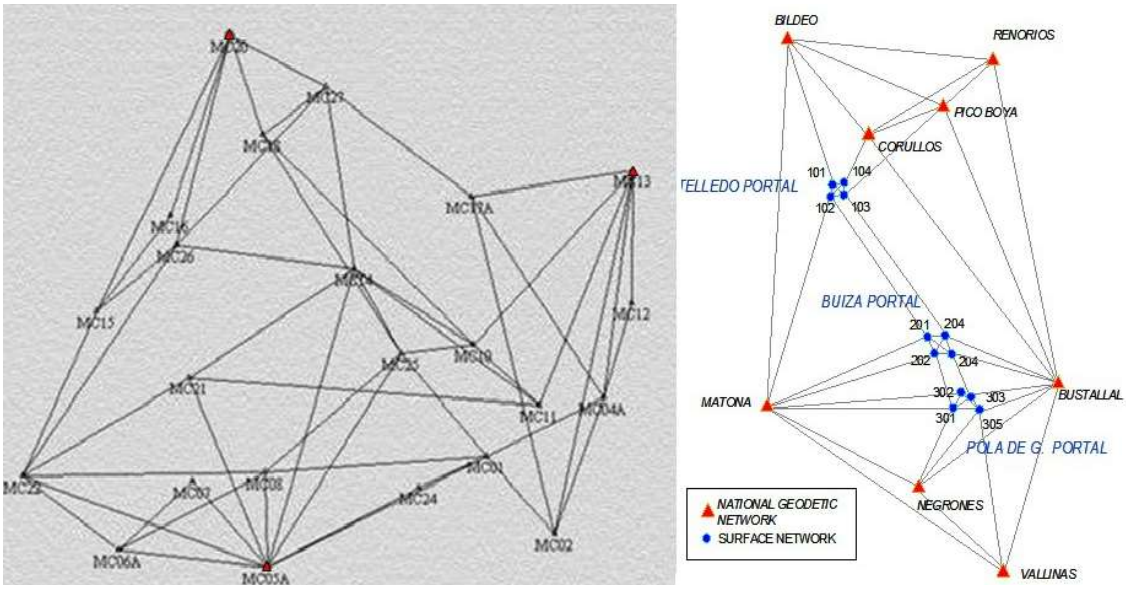



**FIGURA 2.7 – REDE PLANIMÉTRICA (A), REDE PLANIMÉTRICA (B).**

As micro redes planimétricas (MR) são RP's de pequena extensão geralmente definidas num sistema de coordenadas local e próprio para o efeito. No entanto, pode ser usada a rede nacional para transporte de coordenadas para as referências da rede.

Para a rede *A* e *B* da figura 2.7 é visto elevada redundância no número de observações e a diferença entre ter mais pontos de controlo a ser observados entre si. Esta evidência é importante na implementação de uma MR, podendo dar à mesma mais homogeneidade e fiabilidade, principalmente pelo número de pontos de referência estabelecido. O critério de ajustamento das observações da rede é feito recorrendo ao ajuste dos mínimos quadrados (MQ). A equação (2.5) relaciona o azimute de uma direção definida por dois pontos de coordenadas M e P (Coordenadas do plano da MR) que é dada por:

$$Az_{ab} = \tan^{-1}\left(\frac{M_b - M_a}{P_b - P_a}\right) + C + R_0 \qquad (2.5)$$

O valor de C é um valor para corrigir o quadrante correto, o valor é o sinal das operações das funções do numerador e denominador do arco de tangente, estes sinais são dados pela figura 2.8 e a tabela 2.1, e $R_0$ representa o zero do limbo do aparelho.

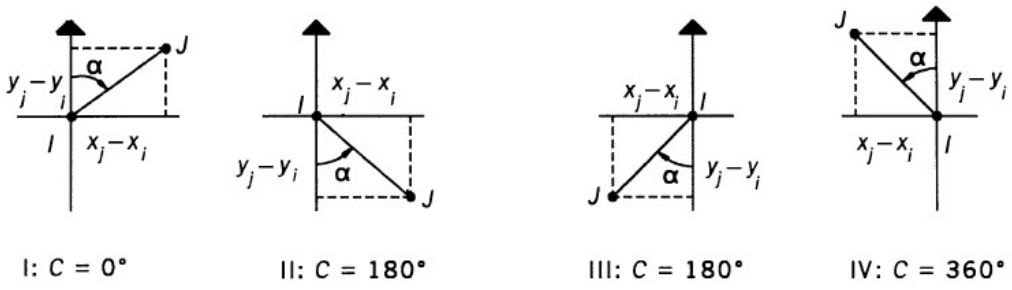



FIGURA 2.8 – RELAÇÃO ENTRE O AZIMUTE AS COORDENADAS DOS PONTOS PARA OBTENÇÃO DO QUADRANTE CORRETO.

| | Sinal de $(M_j - M_i)$ | Sinal de $(P_j - P_i)$ | Sinal de α | C | Azimute Final |
|---|---|---|---|---|---|
| I | + | + | + | 0º | α |
| II | + | - | - | 180º | α + 180º |
| III | - | - | + | 180º | α + 180º |
| IV | - | + | - | 360º | α + 360º |

TABELA 2.1 – TABELA COM RELAÇÕES DA FIGURA 2.8.

A equação da distância observada é dada pela equação 2.6, onde M e P são coordenadas de dois pontos *a* e *b* (dois pontos da $MR$) e D a distância horizontal.

$$D = \sqrt{(M_b - M_a)^2 + (P_b - P_a)^2} \tag{2.6}$$

Relativamente às distâncias é importante referir que a distância medida é a distância inclinada devendo a mesma ser convertida para distância horizontal, tal é possível como mostra a figura 2.8 e a equação 2.7.

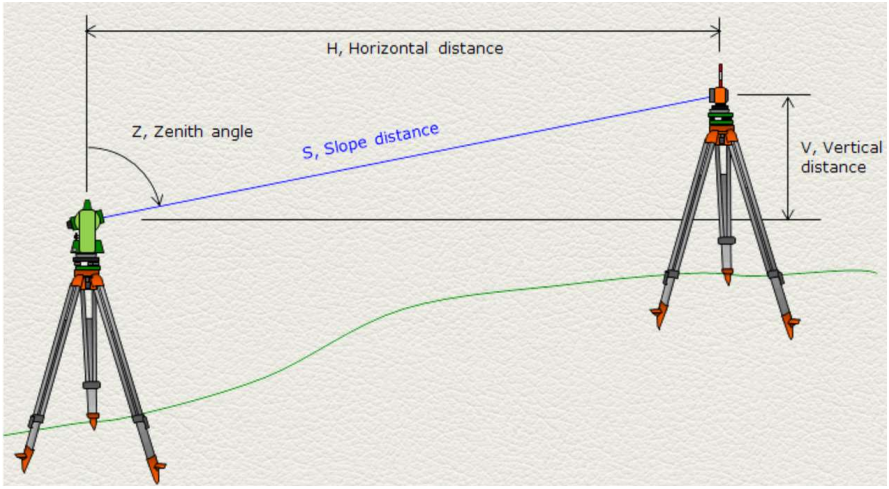



FIGURA 2.8 – RELAÇÃO ENTRE A DISTÂNCIA E O ANGULO ZENITAL DE UMA OBSERVAÇÃO.

A equação 2.7 permite o cálculo da distância horizontal (Horizontal Distance). Onde D é a distância horizontal, S a distância inclinada observada e Z é a observação zenital observada. Isto será importante para trabalhar corretamente no ajustamento das observações e claro os resultados finais, os parâmetros desconhecidos M e P de cada ponto.

$$D = S \sin Z \tag{2.7}$$

Em diversas situações de engenharia é empregue um método chamado “Método por Estação Livre”, que consiste em estacionar a estação total em qualquer lugar perto da rede a observar, sem ter que estar constrangida a ter que estacionar numa referência específica. E isto pode ser sempre empregue em todas as épocas.

Se o projeto consistir em estacionar em referências, então terá que se estar preparado para situações que não permitam observar uma série de alvos da rede, que o foram em épocas anteriores. Pode acontecer que foi montada uma grua ou uma estrutura de apoio de obra que tapou a visão dos alvos, por exemplo, entre outros variados motivos.

A rede de monitorização em 2.9 e 2.10 dispõem de seis pontos a serem monitorizados, duas referências com garantia que não se movem ao longo do tempo, para não introduzir erros em todo o processo e duas estações livres a E1 e a E2.

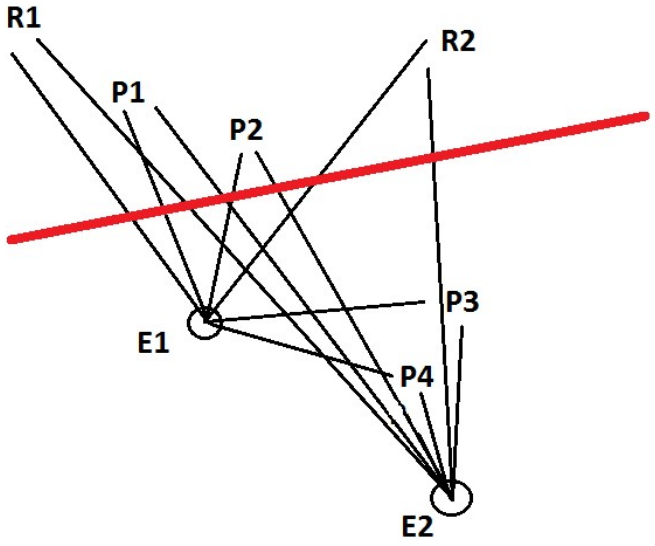



**FIGURA 2.9 – REDE GEODÉSICA DE MONITORIZAÇÃO COM UM OBSTÁCULO NO SEU INTERIOR.**

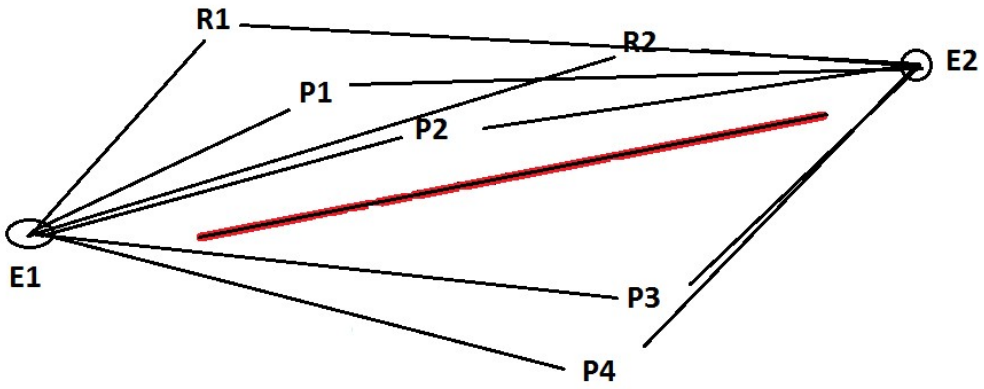



FIGURA 2.10 – REDE GEODÉSICA DE MONITORIZAÇÃO COM UM OBSTÁCULO IMPEDITIVO.

Supondo que o período entre épocas de monitorização é de um mês, e que em determinada época posterior foi permitida a instalação de um obstáculo, simbolizado na figura como o traço vermelho (2.9) o que muda completamente a observação da rede. No entanto, se utilizar o método por estação livre posicionando a estação em outros locais como em (2.10) conseguiria ser observada toda a rede a partir de novos E1 e E2. Para este método é necessário, mas não obrigatório, que as referências se encontrem espaçadas entre si ao longo da estrutura.

# 3 Tratamento da Informação

## 3.1 Algoritmos de ajustamento

Do que já foi referido anteriormente, foram definidas três equações para satisfazer o nivelamento geométrico e as redes planimétricas, e fica claro que a primeira das equações é linear (nivelamento) enquanto as outras duas são não lineares.

Para o processo de ajustamento foram rescritas as equações sob a forma $\mathrm{Ax} = \mathrm{f}$. Com o número de observações ($n_0$) e o número de parâmetros ($n_{m0}$) a serem estimados é calculado o número de graus de liberdade df que é definido da forma (3.1):

$$\mathrm{df} = \mathrm{n}_0 - \mathrm{n}_{\mathrm{mo}} \tag{3.1}$$

Se df for negativo, faltando graus de liberdade, não há modo de resolver sistema de equações, se forem exatamente 0 o sistema tem uma solução se for superior a zero tem múltiplas soluções. Estamos interessados no caso em que df > 0 e na solução que minimiza o quadrado dos resíduos.

O algoritmo de ajustamento é feito pelo critério dos mínimos quadrados, que tem como objetivo a minimização da função da soma quadrática dos resíduos (3.2).

$$\Phi = v^T P_l v \tag{3.2}$$

O primeiro processo é estimar os pontos ajustados (3.3).

$$\begin{gathered} \hat{x} = N^{-1} F_n \\ \hat{x} = (A^T P_l A)^{-1} A^T P_l W \end{gathered} \tag{3.3}$$

Onde $A$ é a matriz de configuração ou coeficientes dos parâmetros, a matriz $P_l$ representa a matriz de peso das observações, a matriz $\hat{x}$ representa os parâmetros de $x$ ajustados, $N^{-1} = (A^T P_l A)^{-1}$ e $W$ a matriz de fecho (ou Ax-f), que representa no caso linear as observações e no caso não linear a equação do modelo funcional.

Para o caso de um modelo paramétrico não linear, é geralmente necessário recorrer-se a um método iterativo para estimar os parâmetros. Quando existir iteração é estabelecido um critério da paragem de iteração, para o caso dos ajustamentos não lineares é de 1E-5 dos resíduos dos parâmetros estimados.

O próximo passo do ajustamento é a determinação dos resíduos das observações (3.4).

$$\hat{v} = A\hat{x} - l \tag{3.4}$$

Com o vetor de resíduos determinado é possível determinar as observações ajustadas (3.5).

$$\hat{l} = l + \hat{v} \tag{3.5}$$

Depois é calculada a variância *à posteriori*, que é necessária para validar o ajustamento com o teste do fator de variância e também para calcular a matriz das estimativas das variâncias e covariâncias dos parâmetros ajustados (3.6).

$$\hat{\sigma_0}^2 = \frac{\hat{v} P_l \hat{v}}{\mathrm{df}} \tag{3.6}$$

O cálculo da matriz das estimativas das variâncias e covariâncias dos parâmetros ajustados terá o seguinte aspeto:

$$\hat{C}_{\hat{x}} = \widehat{\sigma_0}^2 (A^T P_l A)^{-1} \tag{3.7}$$

### 3.1.1 Nivelamento geométrico

A monitorização feita por observações de nivelamento é definida de época em época, ou seja, cada $\mathrm{Eo}$ é ajustada e comparada à época de referência, neste caso a aplicação considera a primeira época como a de referência. Não fazendo, ainda, qualquer outro tipo de possibilidade de calcular a época de referência, como média de duas épocas por exemplo.

Os dados de entrada necessários para um projeto de nivelamento na aplicação observações e respetivas referências. As observações compreendem as seguintes propriedades do ponto:

- Nome dos pontos visados atrás e à frente
- Leituras das réguas atrás e à frente
- Precisões das leituras
- Distâncias entre a régua e o nível
- Altura do instrumento

Neste ajustamento, por $\mathrm{MQ}$ a matriz $A$ tem a forma representada em (3.9), $P_l$ que corresponde à matriz de pesos, é dada pelo inverso da variância das observações de $l$ (3.12). Relembrando a equação (2.4), o modelo funcional (3.8) tem a forma:

$$F(H_i, H_j) = \mathrm{H}_J - \mathrm{H}_i - (l_j - l_i) + v_{ij} \tag{3.8}$$

Ou sob a forma matricial:

$$Ax = l \tag{3.9}$$

Em que o vetor $x$ (3.10) é a altitude de cada um dos pontos, sendo esta matriz o número de pontos ou incógnitas desconhecidas. É assumido que a altitude do ponto 1 (H1) é conhecida.

$$x = \begin{bmatrix} H2 \\ H3 \\ \vdots \\ H_{n-1} \\ H_n \end{bmatrix} \tag{3.10}$$

A matriz $A$ (3.9) é dada pelas derivadas parciais da equação (3.8) em ordem aos parâmetros a estimar, as altitudes. Para este caso particular do nivelamento terá o seguinte aspeto (3.11):

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.11}$$

De notar que cada observação repetida no nivelamento irá traduzir-se numa linha idêntica em $A$, o que será diferente será apenas o peso da observação e a observação em si, no vetor de fecho $W$.

A matriz das variâncias e de observações terá o seguinte aspeto:

$$C_l = \begin{bmatrix} C_{l1} & 0 & 0 & 0 \\ 0 & C_{l2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & C_{li} \end{bmatrix} \qquad l = [l_1 \quad l_2 \quad \cdots \quad l_i] \tag{3.12}$$

Dado que o modelo é linear a matriz $W$ (3.13) será composta pelas observações, a mesma terá o seguinte aspeto:

$$W = \begin{bmatrix} l_2 - l_1 \\ l_3 - l_2 \\ \vdots \\ l_n - l_{n-1} \end{bmatrix} \tag{3.13}$$

O peso definido para cada observação é dado por o inverso do quadrado da precisão do desnível (as variâncias e covariâncias de $C_l$), somado com a precisão do aparelho para cada troço em função da distância (3.14), este peso era o que era utilizado na Geoide®.SA, por isso foi implementado o mesmo:

$$C_l = \sigma_i^2 \ + 0.0015 * \frac{D_i}{1000} \tag{3.14}$$

$0.0015 * \frac{D_i}{1000}$ corresponde à precisão do aparelho, especificada pelo fabricante, para uma linha de nivelamento de 1Km, por isso $D_i$ corresponde ao troço de nivelamento da observação.

### 3.1.2 Micro redes planimétricas

Na monitorização de estruturas é comum existirem pontos de referência, em redor da estrutura e colocados em locais estáveis em termos de deslocamentos, para que se considerem isentos de erro. Estes pontos se fiáveis em termos de deslocamento permitem simplificar o processo de ajustamento da rede.

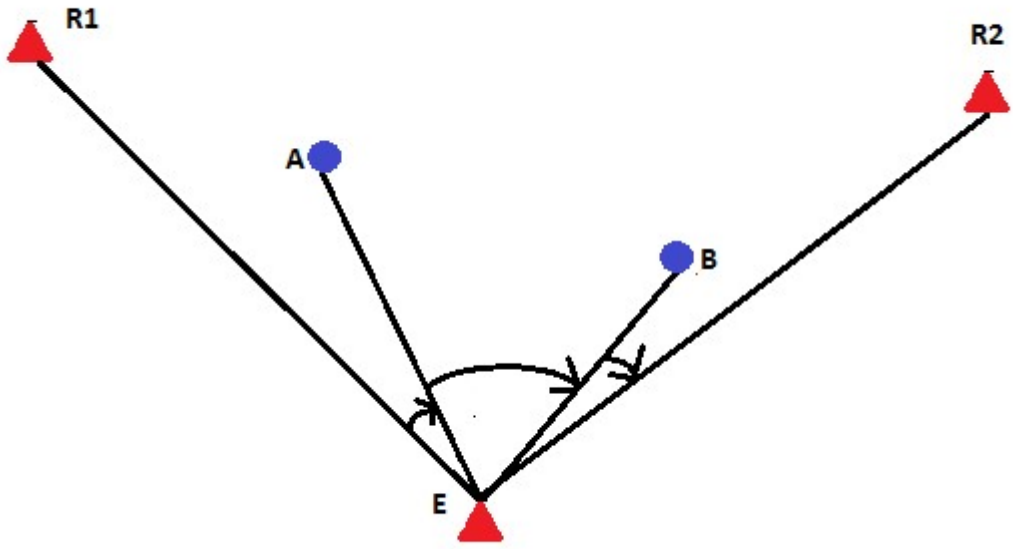



FIGURA 3.1 – REDE DE MONITORIZAÇÃO COM OBSERVAÇÃO DE REFERÊNCIAS.

A diferença dos azimutes das direções A e B indicadas na figura 3.1 é expressa pela equação (3.15) **[1]**:

$$\mathrm{A_{AEB}} = \tan^{-1}\left(\frac{M_A - M_E}{P_A - P_E}\right) - \tan^{-1}\left(\frac{M_B - M_E}{P_B - P_E}\right) + \mathrm{D} \qquad (3.15)$$

E o modelo funcional é dado por:

$$\mathrm{F}(\mathrm{M}_A, \mathrm{P}_A, \mathrm{M}_A, \mathrm{P}_B, \mathrm{M}_E, \mathrm{P}_E) = \tan^{-1}\left(\frac{M_A - M_E}{P_A - P_E}\right) - \tan^{-1}\left(\frac{M_B - M_E}{P_B - P_E}\right) + Az_{EA} - Az_{EB} - \mathrm{D}$$

Como ilustrado na figura 3.1, ao entrar com as referências no processo de observação, é possível a utilização da equação 3.15. Com a observação da direção azimutal, ou azimute, a orientação do limbo $\mathrm{R}_0$ deixa de ser uma incógnita no sistema, passando a não existir cálculo da mesma. $\mathrm{D}$ é a soma de C de cada uma das direções, referido na equação (2.5). A, E e B são os pontos envolvidos na obtenção do azimute pelas duas observações das duas direções, *A e B* a partir da estação *E, M e P* são as coordenadas dos pontos envolvidos.

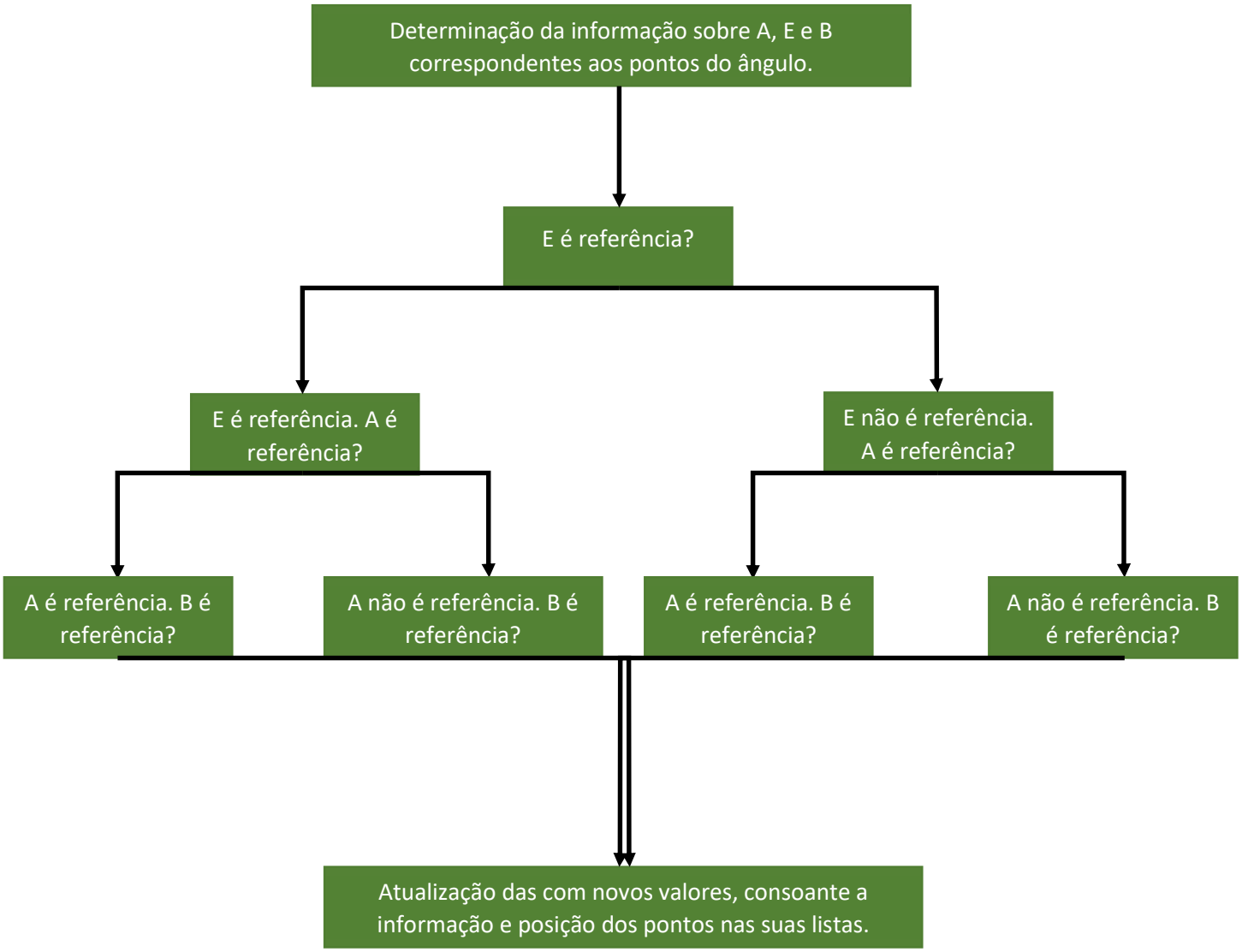
A equação (3.15) é não linear nos parâmetros a determinar pelo que terá de ser linearizada (3.16), obtendo-se a seguinte equação:

$$\begin{aligned}&\left(\frac{P_E-P_B}{IB^2}\right)d_{M_E}+\left(\frac{M_B-M_E}{IB^2}\right)d_{P_E}+\left(\frac{P_B-P_E}{IB^2}\right)d_{M_B}+\left(\frac{M_E-M_B}{IB^2}\right)d_{P_B}\\ &-\left(\frac{P_E-P_A}{IA^2}\right)d_{M_E}+\left(\frac{M_A-M_E}{IA^2}\right)d_{P_E}+\left(\frac{P_A-P_E}{IA^2}\right)d_{M_A}+\left(\frac{M_E-M_A}{IA^2}\right)d_{P_A}\end{aligned} \qquad (3.16)$$

Onde $IA^2$ *e* $IB^2$ (3.17) são dados por:

$$\begin{aligned}IA^2&=(M_A-M_E)^2+(P_A-P_E)^2\\ IB^2&=(M_B-M_E)^2+(P_B-P_E)^2\end{aligned} \qquad (3.17)$$

Com esta informação é necessário construir as matrizes $A\,e\,W$, matriz dos coeficientes das derivadas das observações. Desta forma o algoritmo de construção destas matrizes é dado que explora a informação dos dados relativos à distância definida entre os pontos $A$ e $B$ está definida no esquema (3.1).




**ESQUEMA 3.1** – ESQUEMA COM O ALGORITMO DE DETERMINAÇÃO DOS COEFICIENTES DE A E W PARA OS AZIMUTES.

A matriz dos parâmetros é dada por:

$$x = \begin{bmatrix} dM_1 \\ dP_1 \\ dM_2 \\ dP_2 \\ \vdots \\ \vdots \\ dM_n \\ dP_n \end{bmatrix} \tag{3.18}$$

A matriz de variâncias covariâncias e de observações terá o aspeto:

$$C_l = \begin{bmatrix} C_{l1} & 0 & 0 & 0 \\ 0 & C_{l2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & C_{li} \end{bmatrix} \qquad l = [l_1 \quad l_2 \quad \cdots \quad l_i] \tag{3.19}$$

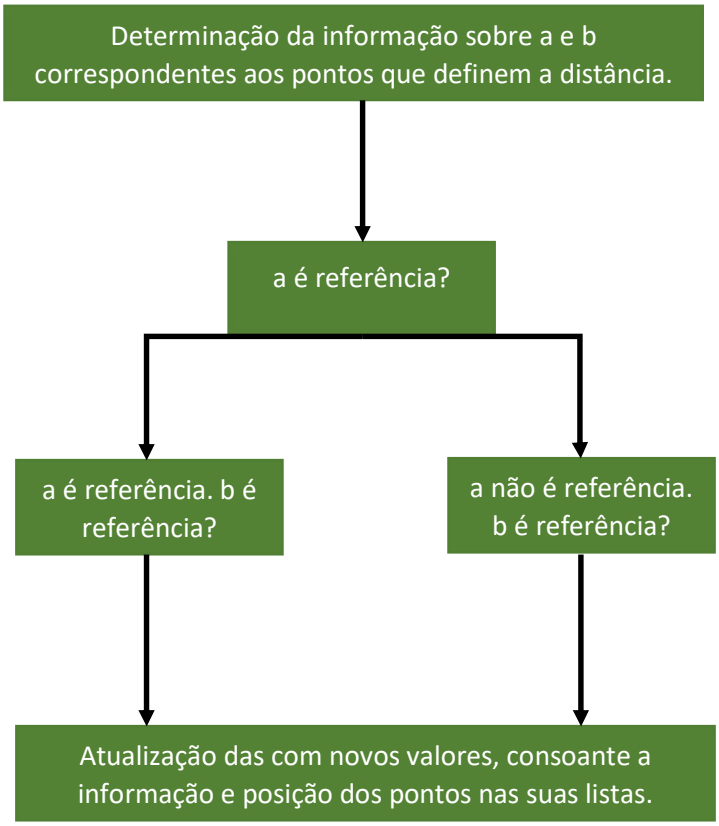
De notar que $C_l$ pode ter outros valores onde existem 0's, sendo a mesma para ir de encontro ao aspeto da matriz $A$.

Dado que o modelo funcional é não linear, a matriz $W$, neste caso, é atualizada com o cálculo da equação segundo os novos valores de $\hat{x}$, os quais foram estimados a cada iteração i, e terá este aspeto:

$$W = \begin{bmatrix} Az_{EA} - Az_{EB} - F()_0 \\ Az_{EC} - Az_{ED} - F()_0 \\ \\ \\ \vdots \\ \vdots \\ \\ \\ Az_{ij} - Az_{ik} - F()_0 \end{bmatrix} \tag{3.20}$$

As variâncias e o vetor de observações são definidos consoante os dados de entrada. As variâncias para a distância são definidos em milímetros e para os ângulos em segundos. Podendo ser segundos de grau (360) ou segundos de grado (400), conforme indicação do utilizador.

O algoritmo que explora o cálculo do ajustamento com as distâncias é mostrado no esquema (3.2).




**ESQUEMA 3.2** – ESQUEMA COM O ALGORITMO DE DETERMINAÇÃO DOS COEFICIENTES DAS MATRIZES A E W PARA AS DISTÂNCIAS.

De acordo com a equação (2.7) obtemos o modelo funcional para as distâncias (3.21)

$$F(\mathrm{M}_A, \mathrm{P}_A, \mathrm{M}_A, \mathrm{P}_B, \mathrm{M}_E, \mathrm{P}_E) = \left(\frac{M_b - M_a}{IJ}\right) dMa + \left(\frac{P_b - P_a}{IJ}\right) dPa + \left(\frac{M_b - M_a}{IJ}\right) dMb + \left(\frac{P_b - P_a}{IJ}\right) dMa - D + r \quad (3.21)$$

A matriz do s parâmetros terá o seguinte aspeto:

$$x = \begin{bmatrix} dM_1 \\ dP_2 \\ dM_2 \\ dP_2 \\ \vdots \\ \vdots \\ dM_n \\ dP_n \end{bmatrix} \quad (3.22)$$

A matriz das variâncias e o vetor de observações teria o seguinte aspeto:

$$C_l = \begin{bmatrix} C_{l1} & 0 & 0 & 0 \\ 0 & C_{l2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & C_{li} \end{bmatrix} \quad l = [l_1 \quad l_2 \quad \cdots \quad l_i] \tag{3.23}$$

## 3.2 Análise de resultados

Após cada ajustamento é feito um teste de razão de variâncias (3.24). Onde a variância *à priori* ${\sigma_0}^2$ é divida pela variância à posteriori ${\sigma_0}^2$. Esta razão depois é dividida pelo inverso do número de graus de liberdade do sistema:

$$y = \frac{\frac{\sigma^2}{{\sigma_0}^2}}{\frac{1}{\mathrm{df}}} \tag{3.24}$$

Este teste apresenta três hipóteses para verificação de rejeição ou aceitação, se uma das hipóteses falhar falhará o teste. As hipóteses são dadas com uma probabilidade com distribuição normal ($X^2$) segundo determinados graus de liberdade $\mathrm{df}$.

Dadas as hipóteses:

$$H_0: \sigma^2 = {\sigma_0}^2 \; VS \; H_1: \sigma^2 \neq {\sigma_0}^2 \tag{3.25}$$

Rejeita-se caso **[8]**:

$$y < {X^2}_{\frac{\alpha^2}{2}}(\mathrm{df}) \; ou \; y > {X^2}_{1-\frac{\alpha^2}{2}}(\mathrm{df}) \tag{3.26}$$

Dadas as hipóteses:

$$H_0: \sigma^2 = {\sigma_0}^2 \; VS \; H_1: \sigma^2 > {\sigma_0}^2 \tag{3.27}$$

Rejeita-se caso **[8]**:

$$y > {X^2}_{1-a} \tag{3.28}$$

Dadas as hipóteses:

$$H_0: \sigma^2 = {\sigma_0}^2 \; VS \; H_1: \sigma^2 < {\sigma_0}^2 \tag{3.29}$$

Rejeita-se caso **[8]**:

$$y < {X_a}^2(\mathrm{df}) \tag{3.30}$$

# 4 Sistema Aplicacional

## 4.1 Introdução

A estrutura da aplicação pode ser implementada de diversas formas e seguindo diferentes metodologias, consoante as necessidades e complexidade da mesma. Isto implica uma escolha de tecnologias que facilitem não só a sua implementação, mas também a sua manutenção, para que no futuro não surjam problemas estruturais, ou seja, problemas que impliquem uma mudança drástica no comportamento da aplicação ou das tecnologias usadas.

Nos últimos dez anos as tecnologias para implementação de aplicações na web aumentaram significativamente, com este aumento surgiram novos conceitos de implementação aplicacional, bem como metodologias que permitem servir melhor essas tecnologias, dando ao programador facilidade de implementação e manutenção das mesmas.

Neste capítulo serão abordados os requisitos tecnológicos necessários para a aplicação. Irá também ser abordada a forma como serão separadas e implementadas as diferentes camadas lógicas da aplicação.

O desenho gráfico da aplicação, ou seja o *front-end,* é também discutido aqui com importância e pormenor, dado que é a interface que o utilizador da aplicação está a ver e a usar, nunca descurando as outras partes da aplicação que o utilizador não vê nem sabe que existem, que é o *back-end*, a qual é a mais importante, como se irá verificar.

O armazenamento de informação será discutido aqui, desde a escolha do tipo de armazenamento, o seu funcionamento e a forma como os dados estão ligados entre si, constituindo uma rede de informação que será explorada pelas tecnologias escolhidas.

## 4.2 Levantamento de requisitos

A aplicação a ser implementada tem como principal objetivo a gestão de projetos de ME. O utilizador tem de poder gerir vários projetos, em que cada projeto tem um conjunto de épocas de monitorização. Tem de permitir guardar todos os dados relativos aos métodos mencionados no capítulo 3. Não existe a possibilidade de um utilizador poder partilhar um projeto com outro utilizador, sendo que isso foi propositado.

O utilizador deve autenticar-se na aplicação com um nome de utilizador, devendo ser o *e-mail*, e uma *password*, no caso do nome do utilizador ser o *e-mail* é enviada informação para o mesmo, como confirmação de registo e alertas.

De seguida o utilizador não tem nada configurado, nem precisa. Está vazio de projetos, podendo criar novos e começar o seu trabalho e utilização da aplicação.

Como a ME é realizada em diferentes épocas, a cada projeto é dada a possibilidade de associar todas as épocas realizadas. Estas épocas por sua vez terão que agregar a sua informação mínima, além desta existem outras complementares, como datas, identificadores e outros dados não tão relevantes para a monitorização, mas para a consistência da informação.

O *front-end* terá que disponibilizar esta informação com a interface já mencionada no objetivo do trabalho, ou seja, uma interface que permita a introdução de dados, visualização de gráficos de deslocamento e resultados, entre outras funcionalidades que serão usadas. Este pormenor irá restringir as possibilidades de escolha da tecnologia a usar, bem como irá implicar a construção de novos *widgets* (componentes gráficos do browser) a serem utilizados.

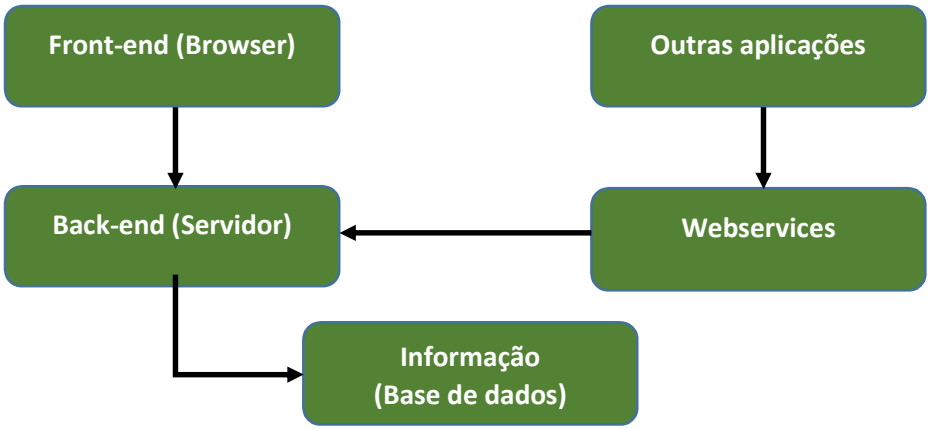
O *back-end* servirá de apoio não só ao *front-end*, mas também a possíveis *Webservices* que poderão ser utilizados por terceiros nas suas aplicações, e permitir também a utilização futura de aplicações móveis com recurso aos mesmos. Isto torna o *back-end* mais complexo do que o *front-end*, onde terá que ser mais cuidadosamente construído.

Como cada utilizador tem uma conta e vários utilizadores estarão a utilizar a aplicação em simultâneo, será necessário ter um gestor de sessões para que o servidor se comporte em paralelo com cada uma delas e não em forma de fila. No entanto, o acesso aos dados, sejam eles por uma base de dados ou acesso a ficheiro, irá ser sempre em fila, não sendo possível contornar este comportamento, apenas sendo possível tentar fazer com que a sua consulta, gravação e alteração sejam feitas da forma mais rápida possível, para libertar o acesso a outro utilizador.

Para que a informação seja acessível e disponibilizada o mais rápido possível de utilizador para utilizador, não se poderá recorrer a ficheiros para armazenar a informação devido à lentidão deste acesso, como tal irá ser utilizado um Sistema de Gestão de Bases de Dados (SGBD) com uma base de dados relacional. Este pormenor é importante, pois apesar de possível é muito complexo relacionar a informação presente em diversos ficheiros dispersos.

A informação tem que ser disponibilizada via *download* ou consulta na interface, dando ao utilizador diversas formas de consulta, não o limitando a escolher apenas uma.

O *back-end* terá que ter os diversos motores que gerem a informação, principalmente os que executam o ajustamento dos dados, pois os mesmos poderão despoletar diversas exceções que têm que ser geridas na sessão e enviadas de volta ao utilizador sobre a forma de texto simples, indicando o erro exato que aconteceu no *back-end*.




**ESQUEMA 4.1** – ESQUEMA DE FUNCIONAMENTO DA ESTRUTURA DA APLICAÇÃO.

Como é possível verificar no esquema 4.1, a estrutura da aplicação funciona em torno do *back-end* que tem acesso aos dados.

## 4.3 Análise, desenho e arquitetura

Depois de definidos os requisitos e tendo a estrutura montada, é necessário analisar mais aprofundadamente o desenho e a arquitetura da mesma. O desenho, neste caso, corresponde somente ao *front-end*, como está organizado e como funciona. A arquitetura é o modo como se irá organizar todo o *back-end* para servir as funcionalidades já descritas. Esta razão é porque ambos estão separados e que comunicam entre si por Remote Procedure Calls (RPC)

As metodologias seguidas em ambos o *front-end* e o *back-end* são diferentes, e o objetivo é manter o código organizado facilitando a implementação e manutenção do mesmo. Nesta altura ainda não são definidas linguagens de programação, apesar de se já ter uma ideia de que analisando os requisitos, as tecnologias disponíveis começam a ficar reduzidas, indicando as possíveis linguagens de programação disponíveis que irão ser usadas.

### 4.3.1 Desenho da interface

A interface deve ser simples e intuitiva de usar. Como referido anteriormente tem que ser rica do ponto de vista dos seus *widgtes*, assemelhando-se a uma aplicação desktop.

Quando comparadas as tecnologias disponíveis para construção de aplicações desktop com interface rica a aplicações web com o mesmo propósito, verifica-se que existe uma diferença muito grande. As linguagens tradicionais para aplicações desktop

trazem consigo bibliotecas já preparadas para o efeito, no que respeita a aplicações web o mesmo não se verifica tão facilmente.

Nos casos das aplicações web, para construir uma interface rica, é necessário conjugar várias linguagens como, HyperText Markup Language (HTML), Cascading Style Sheets (CSS), Hypertext Preprocessor (PHP) e JavaScript (JS). No entanto, ao construir esta aplicação foram evitadas as mesmas de forma direta, pois o Google-Web-Toolkit (GWT) (detalhado mais à frente) é escrito na linguagem Java e converte o código para (JS). Ainda assim, grande parte das aplicações no mercado são concebidas recorrendo a esta conjugação, dado que na altura em que foram implementadas não havia ainda as soluções que se encontram no presente.

Existem vários padrões para aplicações web, foi decidido usar um modelo simples e intuitivo onde se colocam as funcionalidades da aplicação num menu posicionado do lado esquerdo, a criação de entidades num menu superior e toda a informação mutável e respetivos ecrãs ao cento, como mostra a figura (4.1).

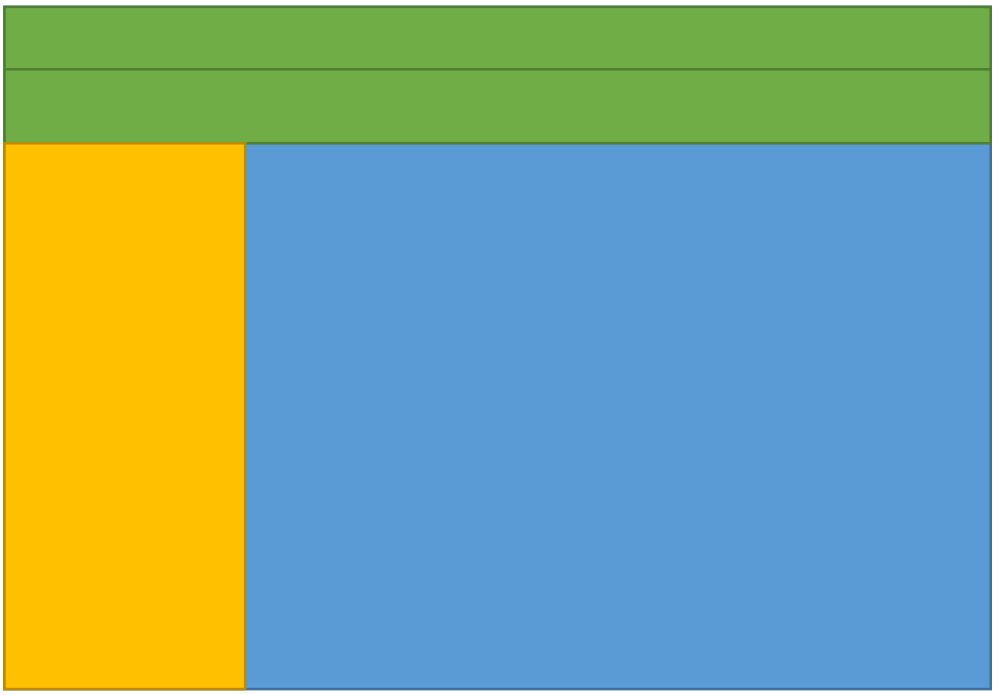


**FIGURA 4.1** – ESQUEMA BASE DO DESENHO DO *FRONT-END*.

Durante a navegação no menu superior (a verde em (4.1)), a sua interface inicial passa por uma página de login, onde o utilizador pode criar a sua conta, recuperar a *password e* efetuar o login. Este menu altera as suas funcionalidades depois de entrar na aplicação.

A navegação no menu lateral (a amarelo em (4.1)) irá despoletar diversos tipos de informação que irão estar disponíveis na parte central da aplicação (a azul em (4.1)). O

tipo de informação que aparece na parte azul será uma página própria (*view*), construída para o efeito.

Durante esta navegação o que na realidade acontece é apenas jogar com a visibilidade ou não das respetivas *view's* criadas e embutidas na componente a azul.


**FIGURA 4.2** – PÁGINA INICIAL DE ARRANQUE DA APLICAÇÃO.

O desenho da interface de *login* do GTweb, na figura (4.2), é composto apenas pelo nome da aplicação, pelos campos relativos ao *login* e uma imagem central.

Após login são apresentados dois menus, um de consulta e outro de criação de informação (4.3). O objetivo desta disposição dos menus influencia a rapidez e simplicidade de como o utilizador pode navegar na aplicação.

Os menus bem como o ambiente de trabalho terão o seguinte aspeto:


**FIGURA 4.3** – PÁGINA DE NAVEGAÇÃO DENTRO DA APLICAÇÃO, COM O MENU LATERAL E SUPERIOR.

Dividindo assim estes dois menus, existe um superior que corresponde à criação de informação, como projetos e épocas de monitorização, e um menu lateral, que corresponde à consulta de toda a informação disponível e armazenada bem como ferramentas. O menu lateral foi implementado de forma a poder ser escondido e mostrado sempre que o utilizador precisar, de modo a que a informação da página seja totalmente visível.

Os botões despoletam ações na página inicial, outras vezes é o próprio programa, que faz com que a componente central mude de aspeto, figura (4.4).

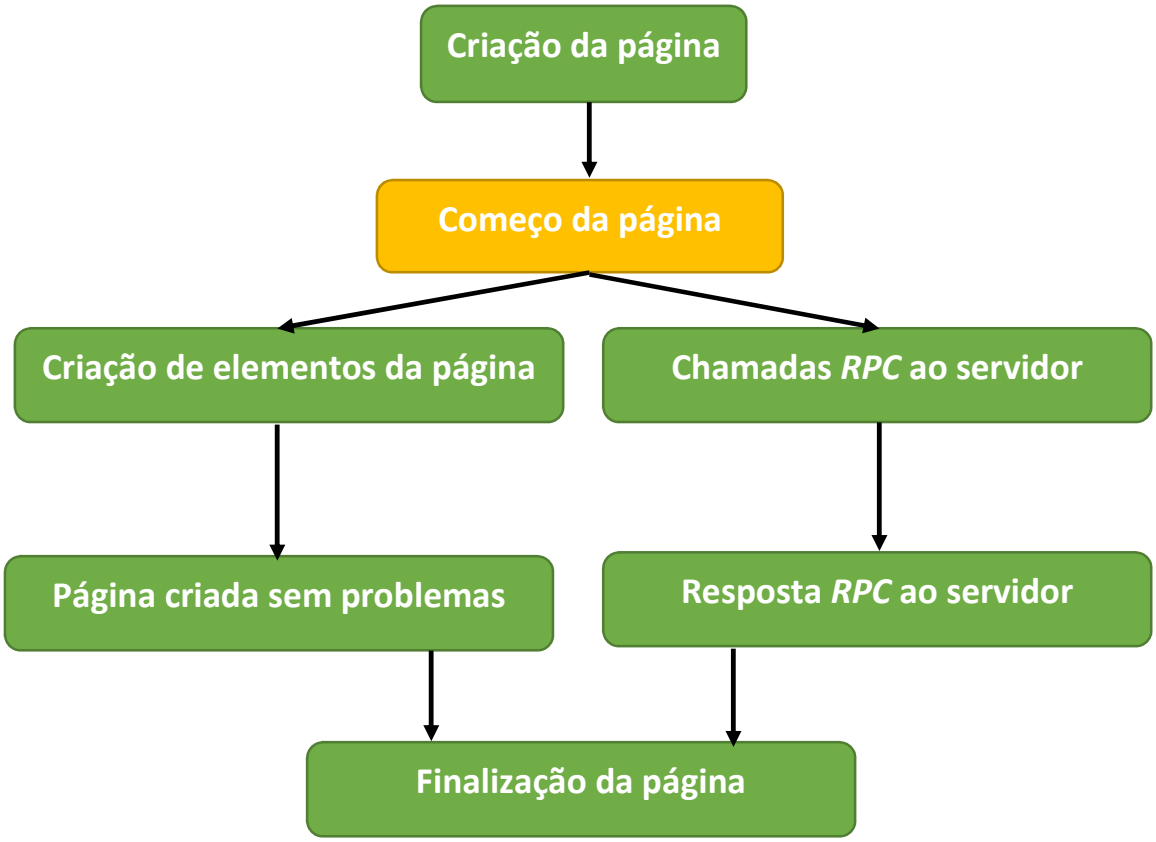

**FIGURA 4.4** – DIVERSOS TIPOS DE INFORMAÇÃO, À ESQUERDA LISTA DE PROJETOS E À DIREITA CONSULTA DOS DADOS DE UMA ÉPOCA DE NIVELAMENTO

As *view's* despoletadas pelo menu lateral podem ainda despoletar outras *sub-view's* e assim sucessivamente. Se o utilizador quer consultar a informação relativa a um projeto, então terá que carregar no botão "Editar", em (4.4) da imagem esquerda, que irá gerar uma nova *view*, podendo novas *sub-view's* seguir-lhe sucessivamente.

Na imagem do lado direito em (4.4) a navegação feita por os botões a azul e verde não despoleta uma nova *view*, mas sim um painel contido na própria *view*. Esta é a diferença entre uma *view* e painéis embutidos numa *view*.

Esta separação deve ser feita com moderação e cuidado, pois implica um custo de memória e também manutenção por parte do programador, porque uma *view* que contenha muitos painéis poderá ter muito código num único ficheiro. Se houver muitas *sub-view's* existirão muitos ficheiros. Por isso deve haver um equilíbrio consoante a necessidade da informação a ser mostrada ao utilizador, devendo cada *view* ser estudada individualmente.

Tem que ser mencionado que durante estes processos de criação ou mudanças de *view's*, pode haver chamadas (RPC) assíncronas ao servidor com o objetivo de trazer a informação invocada para a mesma (4.2). Estas chamadas, por serem assíncronas, nunca garantem que a ordem de chegada da informação do servidor é igual à ordem de chamada pelas *view's.* De modo a estabelecer uma ordem correta, o programador tem que ter em atenção que só após a receção de um pedido ao servidor é que pode realizar um novo pedido ao mesmo.




**ESQUEMA 4.2** – CRIAÇÃO DE UMA PÁGINA ATÉ ESTAR DISPONÍVEL PARA APRESENTAÇÃO.

### 4.3.2 Widgets e outras ferramentas gráficas

Foi mostrado acima que a interface foi construída com base em *widgets* que não existem nas comuns linguagens disponíveis, mas sim a partir de *widgets* básicos e depois aprimorados consoante a necessidade.

A partir de um painel básico de uma linguagem web é possível criar um botão, e a partir de um botão é possível criar um botão mais personalizado, consoante a necessidade de visualização pretendida, como é mostrado em (4.5).


**FIGURA 4.5** – COMPARAÇÃO ENTRE OS *WIDGETS* TRANSFORMADOS A PARTIR DE UM *WIDGET* BÁSICO.

Outros *widgets* necessários à aplicação são gráficos de linhas, que possibilitem a consulta de informação relativa à diferença de deslocamentos entre épocas. Neste caso, não foi necessário construir o gráfico. Existem vários *Webservices* que disponibilizam este tipo de gráficos. Para a aplicação foram usados os gráficos disponibilizados pelos *Webservices* da Google-Charts®.

Desta forma o programador não precisa de construir um gráfico com complexidade, dado que o mesmo já existe disponível e livre de eventuais erros de programação, que poderiam ser cometidos (4.6).

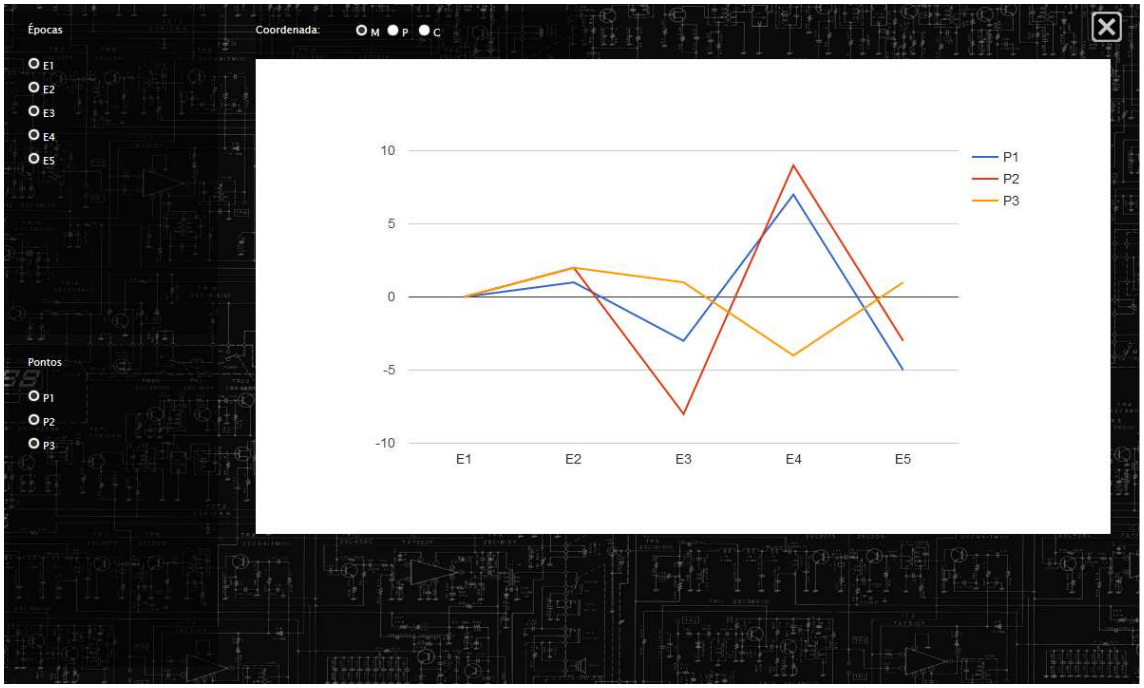



**FIGURA 4.6** – *WIDGET* GRÁFICO DISPONIBILIZADO PELA API DO GOOGLE-CHARTS® PARA COMPARAÇÃO DE DESLOCAMENTOS DOS PONTOS E ÉPOCAS ESCOLHIDOS.

Em todos os painéis, botões ou outros *widgets* que implicam interação do utilizador, existem três ações muito importantes. O botão do rato não pressionado, o mesmo pressionado e por fim pressionado durante um tempo específico. Estas três diferentes ações tornam claro o que o utilizador está prestes a fazer e ajudam-no a navegar pelo menu. Cada uma destas ações tem que ser implementada de raiz pois as mesmas não existem no *widget* base.

Os restantes componentes da aplicação como imagens, caixas de texto ou outros tipos de informação gráfica foram criados a partir de dois componentes apenas, painéis simples e caixas de texto, que foram enriquecidas para se complementarem com o resto dos *widgets* criados.

A criação dos *widgets* é uma tarefa morosa e que envolve bastante tempo, mas uma vez programados, são fáceis e rápidos de alterar, mesmo para um aspeto completamente diferente do anterior. Este facto só é possível devido a métodos de classes que representam o *widget* estão bem definidas e separadas pelo programador.

## 4.4 Metodologias e padrões de arquitetura

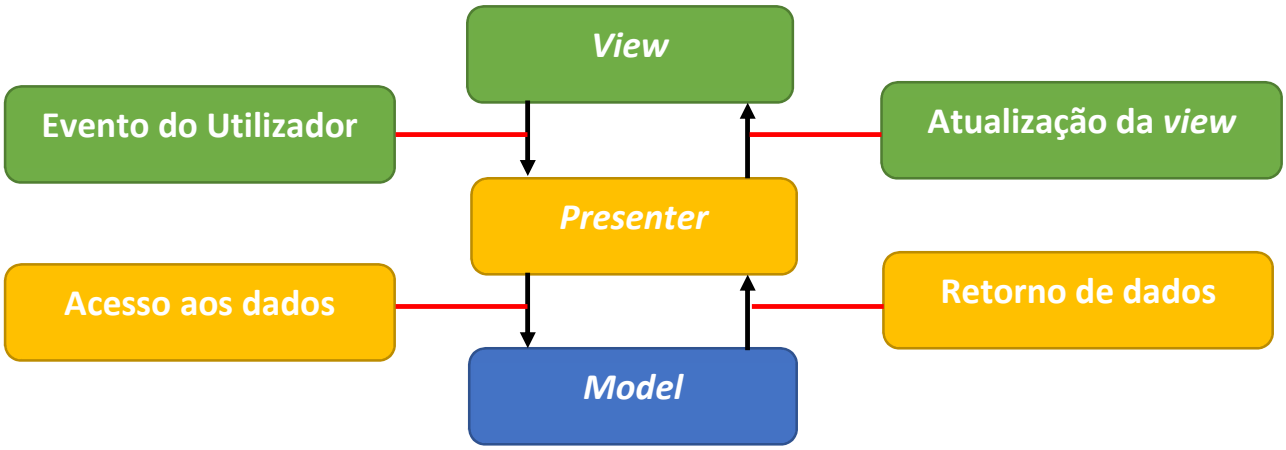
Aplicações mais complexas ou mais abrangentes tendem a ser bastante extensas e por isso complicadas de manter. São definidas muitas vezes entre programadores, estratégias, paradigmas ou padrões de desenho, de modo a facilitar a metodologia de programação da aplicação entre os seus diversos componentes.

Aqui vão ser descritas as metodologias seguidas na implementação do *front-end* e do *back-end*. Estas metodologias têm como principal função separar o código entre as suas diferentes funcionalidades.

A implementação de metodologias de separação de funcionalidades é crucial para a compreensão do código aplicacional bem como a sua manutenção. Existem muitas metodologias a seguir, no entanto para esta aplicação foram apenas necessárias três.

### 4.4.1 Model – View – Presenter

A metodologia Model-View-Presenter (MVP) (4.3), como o nome indica, é uma metodologia que permite à view comunicar com o modelo de dados através de um presenter. Esta metodologia é derivada da metodologia Model-View-Controler (MVC).




**ESQUEMA 4.3** – REPRESENTAÇÃO DO MODEL-VIEW-PRESENTER.

A *view* é a representação visual de uma página do *front-end*. O utilizador quando executa uma ação sobre a página, ou qualquer elemento gráfico da aplicação, vai despoletar uma chamada ao *presenter.* A *view* contém o código de todos os *widgets* e suas funcionalidades, ou seja, a *view* contem, por exemplo, botões onde estão definidas as ações de quando se clica sobre os mesmos. São estas ações que fazem a chamada ao *presenter*.

O *presenter* pode ou não executar uma chamada ao servidor para obtenção de dados que por sua vez irá receber, processar e disponibilizar na *view*. O *presenter* quando invoca o servidor, por exemplo, para obtenção de informação de uma determinada época de monitorização, é feito por um clique em um botão da *view*, em

que este executa um método específico do *presenter*. Este método é um RPC que executa a camada de lógica do *back-end*, retornando para o presenter a informação pretendida.

Se for necessário tratar essa informação antes de a mostrar na *view*, este tratamento é feito no *presenter* e no final passado para a *view*. A *view* tem apenas como função disponibilizar nos seus *widgets* a informação e fazer comportamentos sobre o seu aspeto.

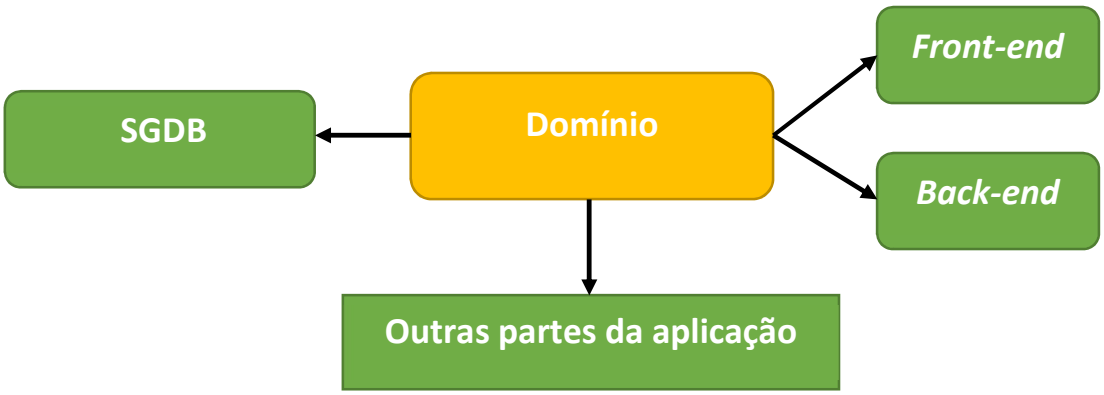
Desta forma a separação de código fica clara, entre o aspeto da página, suas funcionalidades e a lógica de tratamento de informação da mesma.

Desta forma reúnem-se os requisitos pretendidos, inclusive o da manutenção do código por parte do programador.

### 4.4.2 Domínio, Business Component e Data Access Object

O Domínio, a amarelo no esquema (4.4), é o principal conceito de toda a aplicação, porque é centrado nele que está o processo de produção. É um objeto que descreve determinado elemento em concreto, como um utilizador, projeto ou época de monitorização entre outros.

Este objeto é utilizado no *front-end* e no *back-end*, tendo a particularidade de conter a sua estrutura tabelar, ou seja todos os campos de uma tabela de determinada base de dados que estão mapeados no objeto e podem ser consultados em qualquer parte do programa. Por exemplo um objeto época teria todos os campos da tabela épocas incluindo outros que fossem necessários ao programa.




**ESQUEMA 4.4** – REPRESENTAÇÃO DOMÍNIO E ONDE O MESMO É MAIS UTILIZADO E MAPEADO.

A representação do código de um domínio tem de conter indicações da tabela, que representa uma entidade, e a que objeto pertence cada coluna da tabela, que é mostrado no exemplo (4.1).

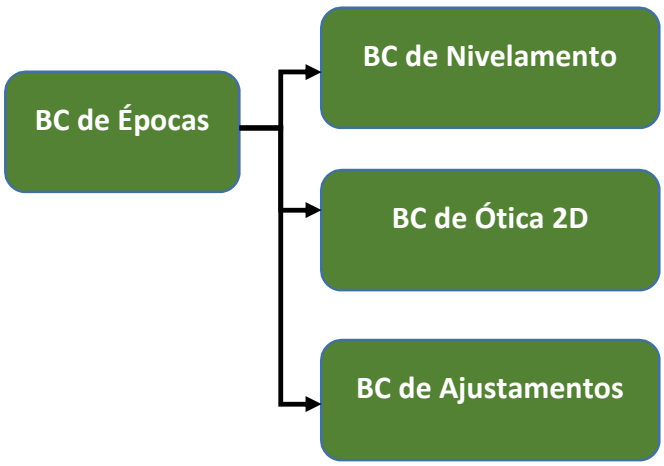
```java
package pt.software.GeoToolsWeb.shared.domains;

import java.io.Serializable;

/**
 * The Class GEOProject.
 */
@Entity
@Table(name = "GEO_PROJECTS")
public class GEOProject extends GEOBase implements Serializable {

    private static final long serialVersionUID = 1364366278260820858L;

    private int id;
    private GEOUser user;
    private String name;
    private int type;
    private boolean active;
    private Date creationDate;
    public GEOProject() {
        super();
    }
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_ID", unique = true)
    public GEOUser getUser() {
        return user;
    }
    public void setUser(GEOUser user) {
        this.user = user;
    }

    @Column(name = "NAME", nullable = false, length = 64)
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Column(name = "TYPE", nullable = false, precision = 1, scale = 0)
    public int getType() {
        return type;
    }
    public void setType(int type) {
        this.type = type;
    }
```

**EXEMPLO 4.1** – REPRESENTAÇÃO DO UM DOMÍNIO, PROJETO NESTE CASO.

O Business Componente (BC) é a camada de lógica de negócio, e é responsável pela gestão de informação, cálculos e toda outra manipulação de dados. Esta camada só está presente no *back-end*, pois não há qualquer vantagem ou interesse deixar este código para o *front-end*.

A separação desta camada com as restantes facilita a sua interpretação por programadores, concentra toda a parte de tratamento de informação e disponibilização da mesma, sem que entre em contacto direto com a utilização da base de dados ou outros serviços de armazenamento.

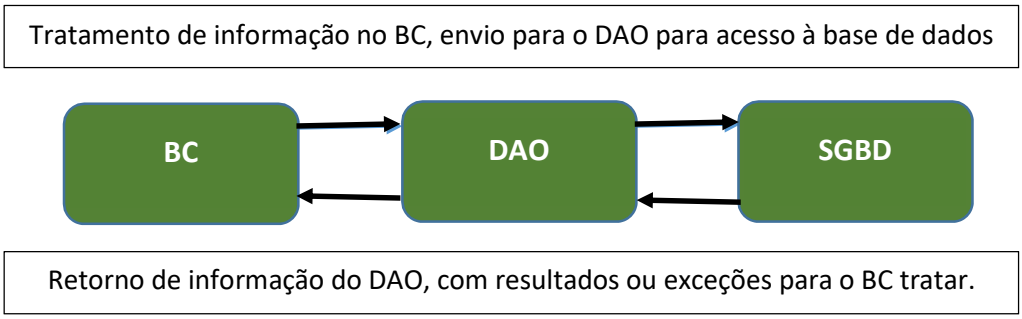
Cada BC pode comunicar com outros BC′$s$ de modo a obter a informação que precisam uns dos outros. Supondo que se pretende repetir o ajustamento de uma determinada época n de nivelamento, é feita uma chamada ao BC das épocas que por sua vez irá consultar o BC de nivelamento e este o BC de ajustamento. A resposta é enviada de volta pelos respetivos BC′$s$ até ao BC das épocas e disponibilizada para o método que o invocou (4.5).




**ESQUEMA 4.5** – REPRESENTAÇÃO DA RELAÇÃO ENTRE DIFERENTES BC'S.

O acesso à informação, que não é feito pelo BC, é assegurado pelo Data Access Object (DAO) onde é possível consultar, armazenar ou alterar algo no SGBD, esta camada recorre sempre a um domínio, onde estão mapeadas as suas propriedades (4.1).

O DAO faz assim operações de leitura, alteração e remoção, atualizando o domínio e a base de dados. O resultado da operação é disponibilizado para quem o pediu. O DAO é assim a única camada que comunica diretamente com o SGBD, tendo como responsabilidade aceder à base de dados e relacionar a mesma com os objetos. É aqui que é feita a leitura, escrita e remoção explícita dos objetos nas respetivas tabelas. Ambas as camadas podem enviar um leque variado de exceções ao utilizador. Podem existir exceções na gravação ou alteração de objetos no DAO, sendo estas enviadas para o BC que o invocou, que as trata e envia de volta para o *front-end* para serem vistas por o utilizador (4.6).




**ESQUEMA 4.6** – REPRESENTAÇÃO DA RELAÇÃO ENTRE BC, DAO E A BASE DE DADOS.

## 4.5 Modelo de dados

O modelo de dados da aplicação é um modelo relacional, permitindo que os dados possam ser armazenados em SGBD's relacionais.

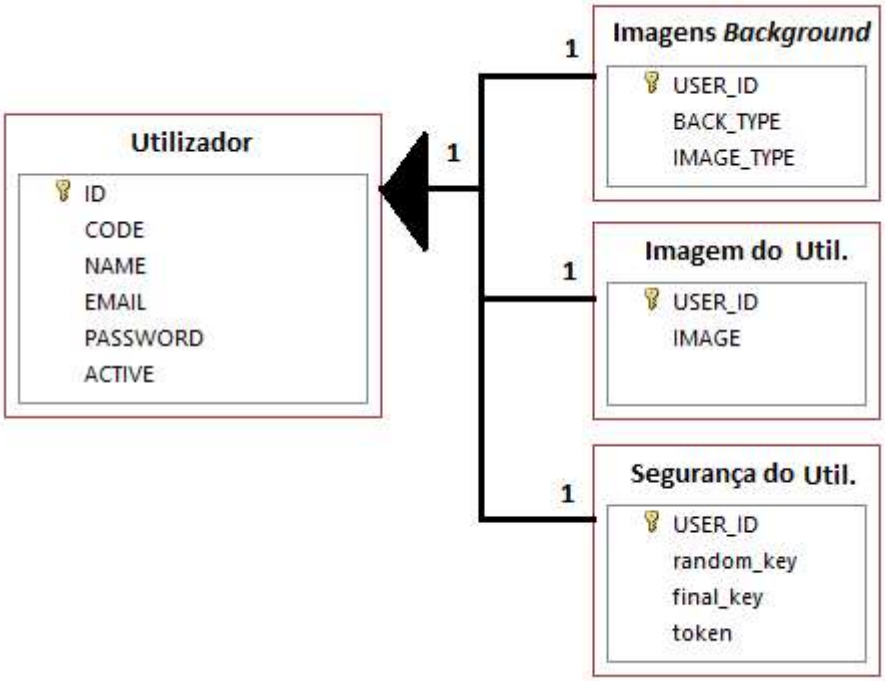
### 4.5.1 Principais entidades e relações

Como os domínios relacionam o que está mapeado na base de dados, vão ser expostos aqui algumas das principais entidades e de como as mesmas se relacionam.

Nesta aplicação o utilizador é tomado como a principal entidade e é em redor desta que se formam as principais relações diretas aos dados. Esta escolha deveu-se principalmente ao facto de ter tomado como obrigatório a existência de um utilizador autenticado, para utilização das ferramentas da aplicação, garantindo sempre um identificador único em cada sessão ao servidor.

Os utilizadores usam uma autenticação que necessita de ser protegida e encriptada, para tal é usado um algoritmo próprio para o efeito, de modo a guardar os respetivos *tokens* de autenticação na base de dados. O *token* é gerado mediante uma ponderação de *strings* com informação do tempo do pedido no servidor, do valor das chaves do utilizador e da data de login.

Esta relação é demonstrada no esquema UML (4.7):



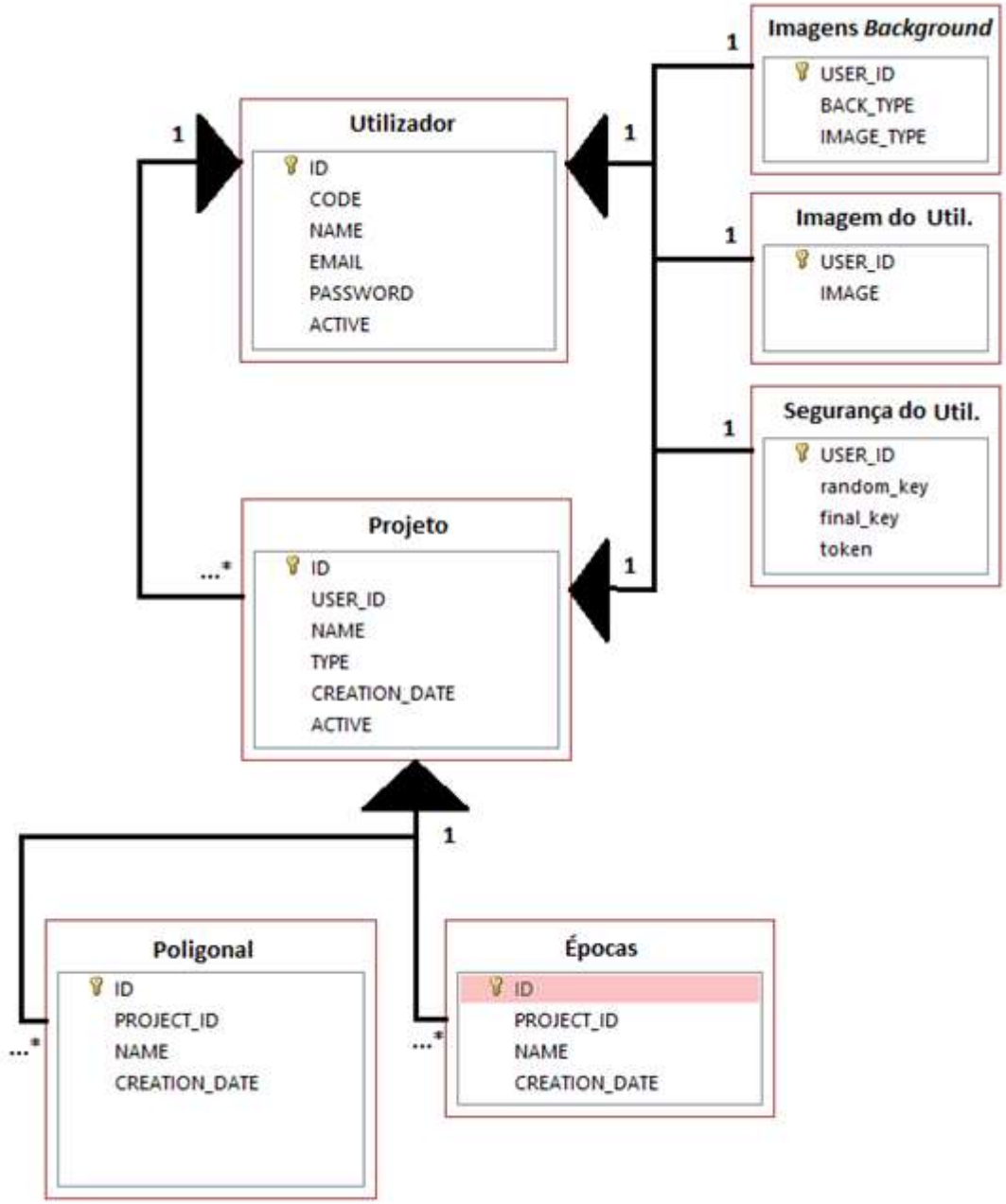

**ESQUEMA 4.7** – RELAÇÃO ENTRE O UTILIZADOR E AS PROPRIEDADES DE AUTENTICAÇÃO.

A principal entidade, Utilizador, só para poder existir na aplicação tem que ter três relações no esquema (4.7). A primeira, Imagem do utilizador, mesmo que vazia, a segunda são as imagens de fundo da aplicação associadas ao utilizador entidade, e por fim a Segurança do utilizador que corresponde aos primeiros *tokens* a serem usados.

As chaves são correspondentes ao algoritmo de criptografia RSA, assim chamado devido aos nomes dos três professores do MIT que o inventaram, Ronald **R**ivest, Adi **S**hamir e Leonard **A**dleman. Com este tipo de encriptação existem duas chaves, a pública e a privada. Quem tiver a chave pública consegue encriptar mensagens, quem tiver a chave privada consegue, além de encriptar mensagens, desencriptar mensagens.

A criação de um projeto de monitorização fica automaticamente ligado a um utilizador, sendo depois este projeto de monitorização que contém toda a informação relacionada com as épocas de monitorização que nele sejam criadas.

Para uma Época ou para uma nova poligonal são necessárias novas relações. Essas relações pertencem a um Projeto como mostra abaixo o esquema UML (4.8):

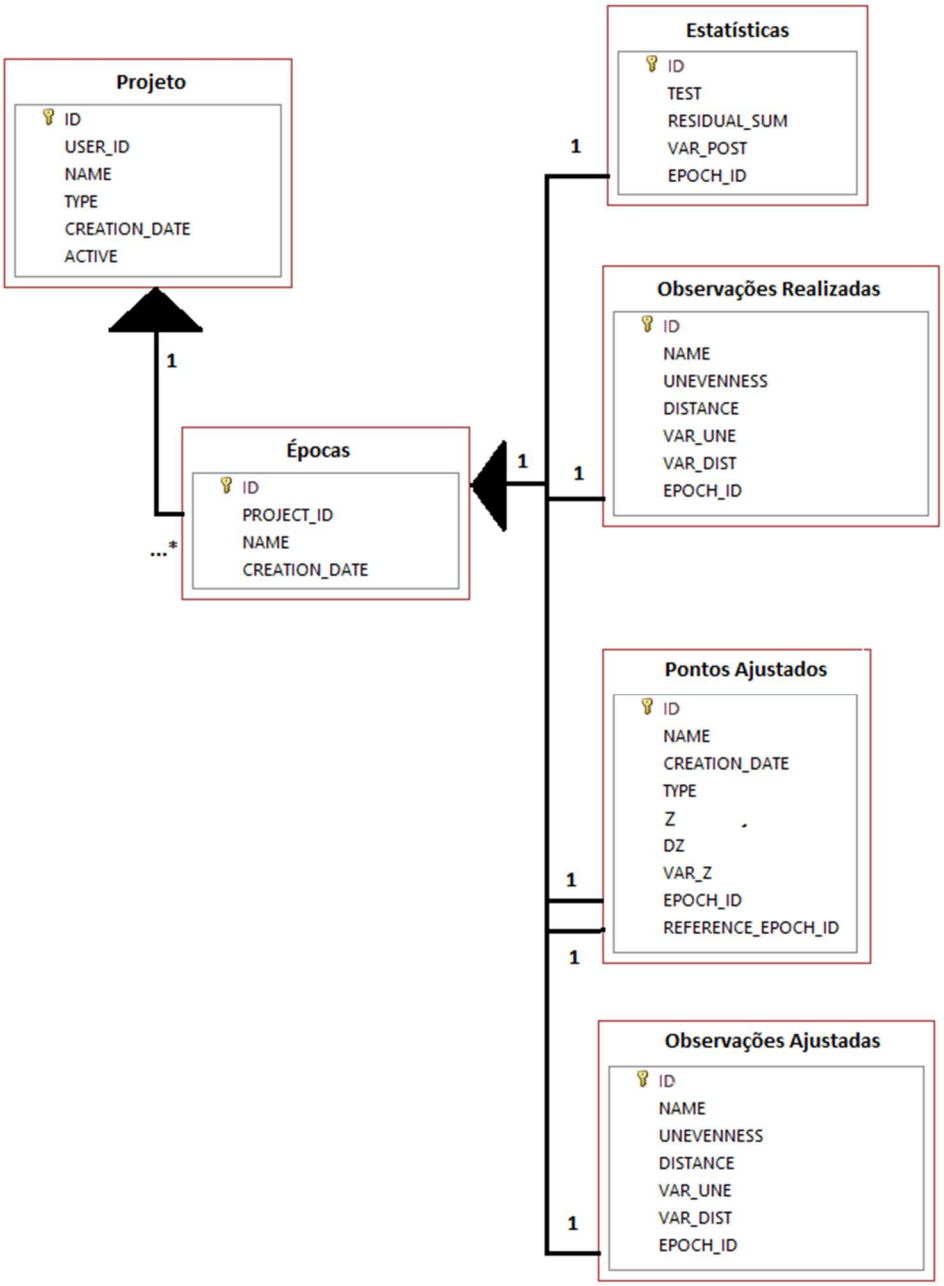



**ESQUEMA 4.8** – RELAÇÃO ENTRE O UTILIZADOR, SEUS DADOS, SEUS PROJETOS E ÉPOCAS DE MONITORIZAÇÃO.

A tabela de poligonal está de parte apenas porque este tipo de poligonal recorre ao ajustamento clássico tratado em topografia e não aos mínimos quadrados, apesar de poderem ser utilizadas os dois tipos para o mesmo fim na aplicação.

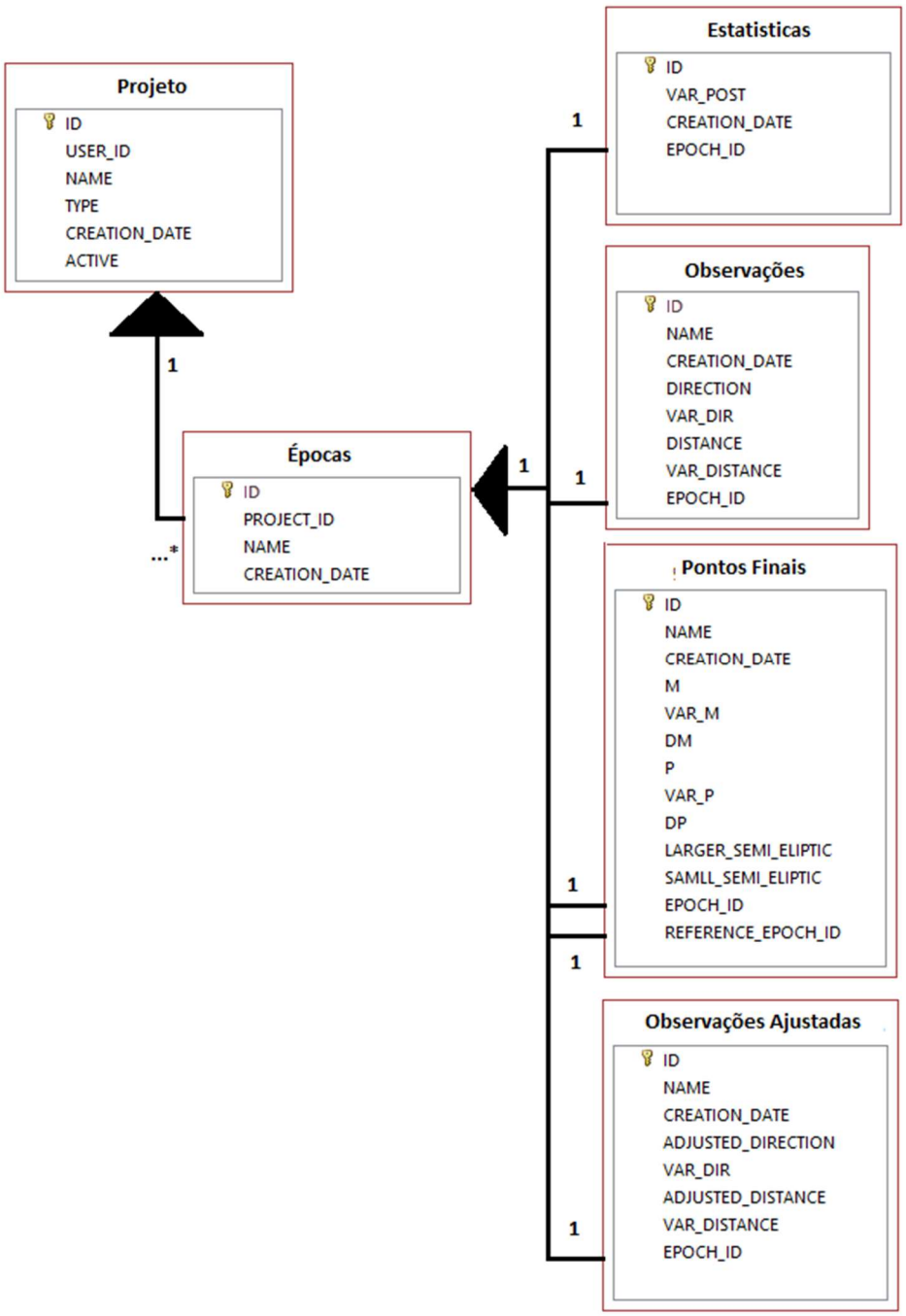
Para um projeto de nivelamento são necessárias pelo menos quatro tabelas (4.9), uma com os dados de observações, dados de observações ajustadas, outra com os pontos ajustados e uma com as estatísticas do ajustamento. No que respeita a outros dados do ajustamento, como matrizes e outros resultados de ajustamento estão

guardados em ficheiros do tipo Binary Large Object (BLOB), devido à dimensão matricial nunca ser prevista após criação da tabela.




**ESQUEMA 4.9** – RELAÇÃO ENTRE UMA ÉPOCA DE NIVELAMENTO E SUAS PRINCIPAIS ENTIDADES.

Da mesma forma para o ajustamento de uma rede planimétrica existem as mesmas tabelas, mas com colunas claramente diferentes (4.10).




**ESQUEMA 4.10** – RELAÇÃO ENTRE UMA ÉPOCA DE PLANIMETRIA E SUAS PRINCIPAIS ENTIDADES.

# 5 Implementação e Tecnologias

Com o objetivo de produzir uma aplicação com os requisitos mencionados, foi necessário procurar uma tecnologia base, que sirva principalmente o *front-end* e o *back-end* da forma descrita anteriormente.

Dado que são várias as tecnologias possíveis de ser usadas para atingir objetivo, ficou decidido que seria utilizado o Google Web *Toolkit* (GWT) que abrange os diversos requisitos mencionados no capítulo 4. Esta tecnologia é escrita em Java, o que permitirá usar outras tecnologias de auxílio no *back-end* também em Java, facilitando todo o processo de *back-end* já definido, incluindo os *Webservices*

Definida a linguagem de programação foi possível utilizar uma tecnologia, baseada em Java, para gerir as sessões dos diversos utilizadores, o Spring. Com esta tecnologia as RPC's ficaram muito mais facilitadas e possíveis de controlar em todas as etapas do *back-end*.

Ficou também facilitado o acesso à informação armazenada na base de dados, devido ao uso da tecnologia Hibernate, que permite um acesso muito mais simples e direto aos dados. Isto porque os domínios têm a estrutura tabelar de todas as tabelas do SGBD.

Desta forma a aplicação poderá ser mais facilmente implementada, bem como num tempo mais reduzido do que se fossem escolhidas outras tecnologias, já que o Spring e o Hibernate não se encontram escritos em outras linguagens de programação.

## 5.1 Java

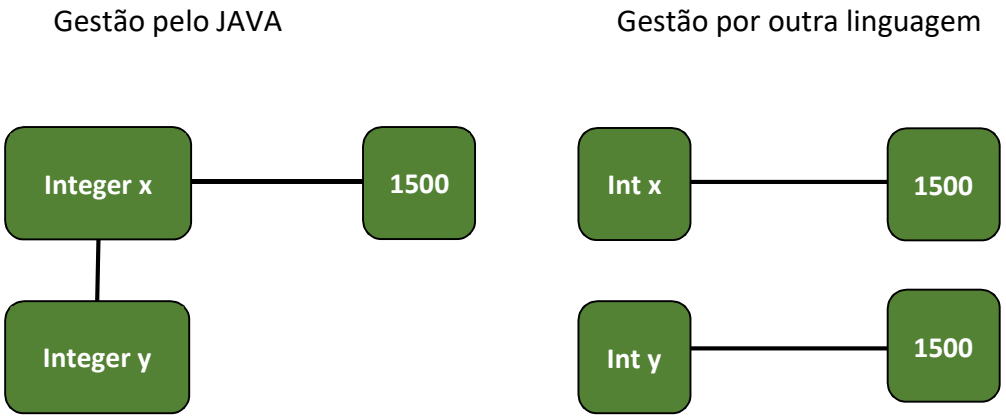
O Java foi a linguagem de programação utilizada. A sua escolha foi condicionada pela escolha da linguagem do GWT, o Java e o GWT tornam-se assim a principal escolha. No entanto, foi também possível reunir uma outra série de tecnologias e bibliotecas, de auxílio à implementação dos motores de ajustamento e gestão de outra informação.

Nesta linguagem o código escrito em Java é compilado para *bytecode* e executado na máquina virtual do Java, garantido desta forma independência da máquina. O mesmo programa pode correr em qualquer sistema operativo com o JRE instalado (Portabilidade).

O Java trabalha com o conceito de objeto, o que quando comparado a outras linguagens de programação, o mesmo não existe, com exceção para algumas mais modernas. O Java permite uma gestão de memória mais eficaz sem estar ao cargo do programador, dado que a mesma é feita pelo *Garbage Collector* Gc.

No esquema 5.1 é ilustrado um exemplo do uso de objetos em que são usadas duas variáveis com valor de inicialização de 1500 (Objeto Integer) para x, sendo y uma

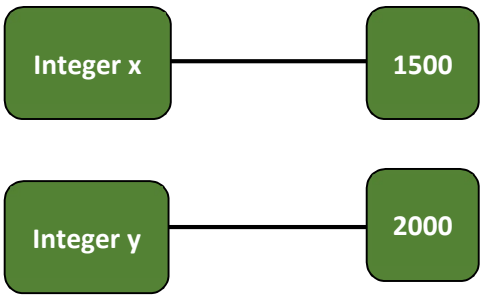
cópia de x. Quando comparado o uso da memória pela parte do Java e por parte de outras linguagens, é possível verificar o uso de menos memória no primeiro caso.




ESQUEMA 5.1 – COMO É GERIDA A MEMÓRIA EM JAVA EM OUTRAS LINGUAGENS

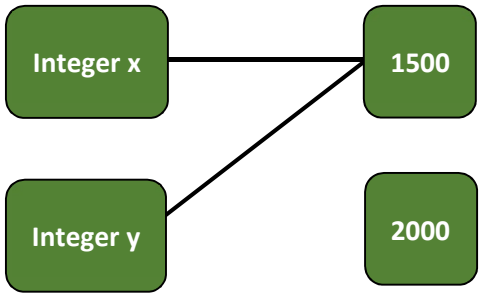
O Java não vai alocar memória para todas as variáveis que o programador cria, copia ou atribui, vai sim criar um apontador para a variável que é exatamente igual, pertence ao mesmo objeto. Poderemos dizer que no caso acima a memória utilizada é quase metade, devido a outros fatores não explorados aqui.

Supondo agora que y em Java é alterado para 2000 (Objeto Integer). O que aconteceria seria o resultado abaixo:




ESQUEMA 5.2 – COMO É GERIDA A MEMÓRIA EM JAVA QUANDO MUDOU O VALOR DE Y

De seguida seria reposto outra vez para 1500, ou seja dizer que é igual a x, o resultado seria o abaixo:




ESQUEMA 5.3 – COMO É GERIDA A MEMÓRIA EM JAVA QUANDO MUDOU O VALOR DE Y

O apontador mudou, mas ficou memória alocada que o Gc pode remover, sendo essa gestão a seu cargo, e não do utilização, apesar de o poder invocar no código.

A linguagem que se pode afirmar que é quase igual ao Java é o C#, também esta definida por objetos, ainda assim é preferível o Java por dispor de mais bibliotecas, construídas por terceiros ao longo dos anos, do que no C#. Os programadores levam muito a sério a existência de livros, documentação, bibliotecas e principalmente milhares de problemas resolvidos. Isto pode ser verificado nos resultados de pesquisas no Google para os mais diversos problemas, descritos em sítios da internet, fóruns, blogues e até redes sociais. Neste aspeto a documentação o C# é significativamente mais pequena em relação ao Java.

As bibliotecas que existem para o Java são muitas, com soluções para quase tudo, neste aspeto assemelha-se muito ao C++. Por exemplo a biblioteca que utilizei para cálculo matricial foi a "Jama", que é praticamente baseada em funções do próprio Matlab, mas tinha mais uma série de bibliotecas que diferiam um pouco entre elas. O C++ também incorpora este tipo de bibliotecas, o C++ foi, e é, uma linguagem muito usada ao longo do tempo e como tal tem um vasto leque de soluções, mas não utiliza o conceito de objeto.

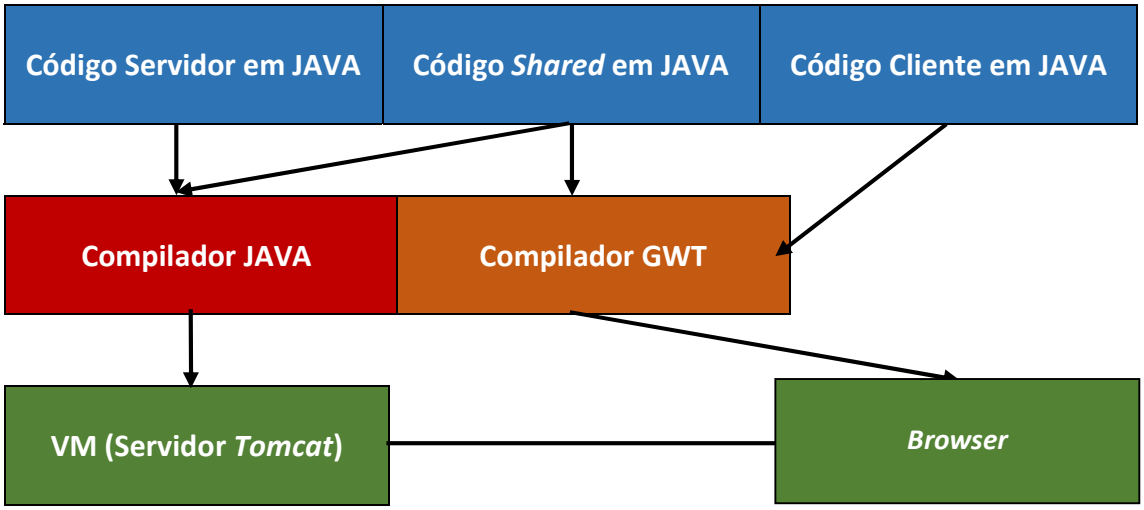
O Java também tem as suas desvantagens, é ligeiramente mais lento a arrancar devido à sua dependência da máquina virtual, e pela mesma razão da existência da máquina virtual o programa comporta-se um pouco mais devagar, porque tem que todo o seu *bytecode* terá que ser transformado para código que a máquina real saiba interpretar.

## 5.2 Google Web Toolkit

O GWT, principal *framework* da aplicação, é um *Toolkit* que permite criar aplicativos web em JS com tecnologia Ajax, mas escritos em Java.

Na base deste *Toolkit* está o conceito de cliente-servidor (*Front-end* / *Back-end*). Devido à separação entre ambos, o código escrito é compilado para duas linguagens diferentes, criando assim as duas camadas de código, a de servidor em Java e a de cliente, a ser interpretada pelo navegador web, em JS.

Esta compilação de código de Java para JS é assegurada pelo próprio *Toolkit*, como o Java tem bibliotecas diferentes, ou inexistentes em JS o mesmo só consegue compilar algumas destas para o *front-end* (java.lang.* e java.util.*), ou seja, classes que sejam serializáveis. Isto implica que o código que é utilizado para programar a componente cliente tem que ser assegurado pela *Toolkit* caso contrário o GWT não o reconhece e não o consegue compilar para JS, como é o caso das bibliotecas nativas I/O ou JPA do Java.




ESQUEMA 5.4 – ESQUEMA SIMPLIFICADO DO FUNCIONAMENTO DO GWT

Como pode ser visto no esquema 5.4 existe uma terceira camada de código, o *shared* (a azul no meio em 5.4), que é código que é partilhado pelo cliente e pelo servidor. Desta forma vão existir três componentes distintas. Uma primeira componente, o *Server-Side* que representa todo o código criado para operar do lado do servidor, o *Client-Side* que representa todo o código a servir o cliente, e a combinação de código que é igual para ambas as componentes, servidor e cliente, o *Shared*, servindo principalmente para evitar manutenção de código duplicado. É preciso especial atenção da forma como se organiza o código do *Shared*, devido ao uso limitativo de bibliotecas, aquelas que o GWT não reconhece não poderão ser usadas aqui, pois irão ser compiladas para JS também.

O GWT disponibiliza também um conjunto alargado de componentes gráficos para construção de interfaces gráficas, os *widgets*. Estes componentes permitem a criação de interfaces apelativas visualmente, complexas e que permitem responder de forma semelhante à maioria das aplicações desktop ou *standalone*. A popularidade do GWT aliada ao facto ser escrita Java, amplamente utilizado, permite uma elevada contribuição da comunidade de programadores, levando ao aumento da variedade de componentes e suas funcionalidades. Isto é um ponto-chave, com vantagem para a criação e manutenção de aplicações de grande dimensão.

É tentador afirmar que com o GWT o programador não tem que preocupar com a interface gráfica nos diferentes *browsers*, porque isso fica ao cuidado do compilador. Esta afirmação não é inteiramente verdade, pois apesar de o GWT fazer a compilação do código para os diferentes *browsers* de forma a manter o mesmo comportamento entre todos, o programador tem seguir as boas regras do GWT. Por diversas vezes somos confrontados com situações que exigem uma solução mais complexa, fazendo com que desta forma possamos ter que seguir uma solução que possa ter comportamentos adversos entre os diferentes *browsers*, ou ser limitada em alguns, estas situações são mãos comuns na construção dos *widgets* e suas funcionalidades. As atualizações dos *browsers* também podem implicar problemas no comportamento da aplicação.

Esta aplicação exigiu algumas soluções pouco comuns para comportar algumas das suas funcionalidades, apesar de nenhum *browser* ter ficado comprometido é possível encontrar alguns pontos visuais diferentes entre eles.

Outra vantagem importante do GWT é que permite o *debugging* do *cliente-side* em ambiente desenvolvimento Java. Para tal existe um módulo disponibilizado chamado Development Mode (Dev Mode). Este modo permite aos programadores executarem código cliente do GWT numa JVM, ou seja o código cliente não é compilado para JS como seria esperado. Isto representa uma vantagem enorme para os programadores, pois desta forma o programador consegue seguir todo o fluxo de código em curso desde o cliente ao servidor como se de uma única linguagem de programação se tratasse.

Por fim de relembrar que os *widgets* foram construídos e aprimorados a partir dos de base com este *toolkit*, que os converte para JS, por isso recorrendo a código java deste que os mesmos foram programados.

## 5.3 Spring

O Spring é uma *framework* aplicacional, escrita em Java, baseado em inversão de controle. No *container* do Spring são executados diversos processos, que caberiam ao programador gerir no código, como:

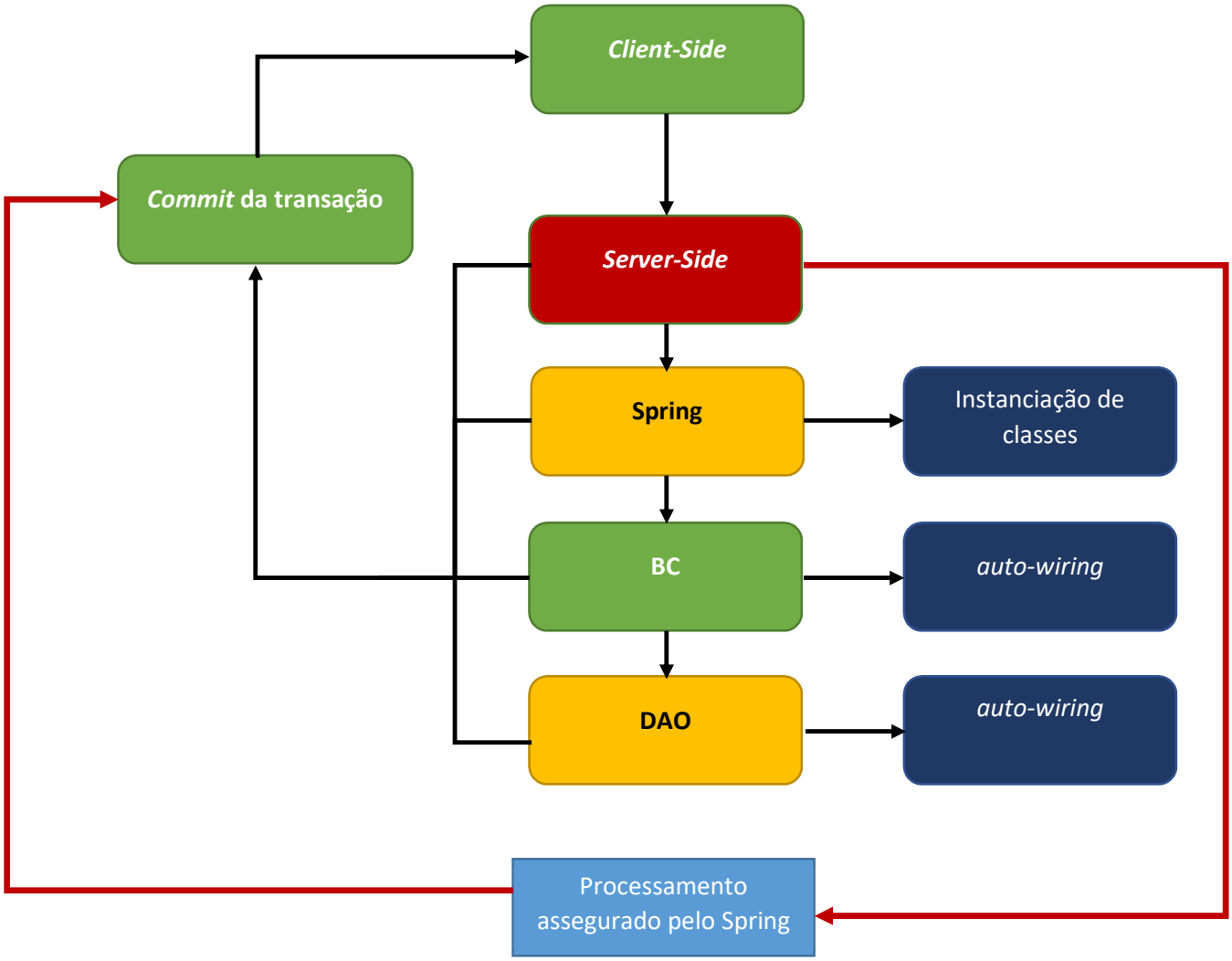
- Instanciação de classes de uma determinada aplicação Java, com vista a ser acessíveis após o arranque da aplicação
- Processo de *auto-wiring*, ou seja, a capacidade de cada classe ser instanciada uma única vez, ficando disponível entre classes
- Controlo de transações, desde a sua criação até à sua destruição

O objetivo do Spring é retirar ao programador toda uma gestão de processos, que de outra forma ficariam muito mais complexos de controlar. O Spring é também uma *framework* com um alto desempenho aplicacional.

Sendo o GWT a base da aplicação, com conceito de cliente e servidor, podemos usufruir do Spring para gerir toda a transação a ser efetuada pelo servidor, inclusive a transação a ser efetuada à base de dados.

O conceito de *auto-wiring* é extremamente importante e dado pelo Spring, que é a instanciação de todas as classes necessárias ao processamento da transação, bem como a disponibilização por dos diversos recursos da aplicação.

O *auto-wiring* é um processo no qual, quando arranca a aplicação no servidor, são inicializadas todas as classes previstas nesse processo. Esta inicialização é feita somente uma vez e disponibilizada para todas as transações, o que significa que não há duplicação de instanciação de classes. Sem o uso do Spring seria necessário instanciar todas as classes a consoante a sua utilização.




ESQUEMA 5.5 – SEQUENCIAMENTO DO TRATAMENTO DE UM PEDIDO PELO SPRING NO SERVIDOR

No esquema 5.5, podemos verificar o comportamento da Spring durante toda a transação. No caso de sucesso do pedido ao servidor, a resposta é devolvida ao cliente com o pedido que o mesmo fez. Se houver um erro no processo de transação, como uma exceção prevista ou a tentativa de corromper a estrutura de uma tabela, toda a transação sofre um *rollback*, ou seja, nada fica alterado e o utilizador recebe uma mensagem previamente programada com indicação do erro.

Os *rollback's* são importantes, pois significa que nada ficou comprometido, deixando para outros utilizadores e sessões exatamente o mesmo estado do que se aquele pedido não tivesse existido.

O uso do *auto-wiring*, como já foi explicado acima, permite que o programador não tenha que se preocupar com a instanciação para que seja possível a utilização de um repositório, sendo esta classe anotada com "*@Autowired*" para o efeito no exemplo (5.1).

Os métodos envolvidos nas transações do Spring são executados consoante a anotação correspondente (5.1). No caso *"@GeoTransacionalReadOnly*", uma transação que apenas executa leitura na base de dados sem fazer qualquer alteração, ou *"@GeoTransacional*" que efetua uma alteração à base de dados. A variável "*usersRepository*" representa uma classe que em modo de acesso por "*auto-wiring*" (@Autowired).

```
package pt.software.GeoToolsWeb.server.Users;

import java.io.UnsupportedEncodingException;

@Service("tmiUsersManagementService")
public class UsersBC implements IUsersBCPrivate {

    @Autowired
    private IUsersDao usersRepository;

    private static final String TASK_ME_IN_KEY = "j8&1";

    @Override
    @TMITransactional
    public GEOUserDefinitions updateUserDefinitions(GEOUserDefinitions user
        try {
            return usersRepository.updateUserDefinitions(userDefinitions);
        } catch (RuntimeException re) {
            throw re;
        }
    }


    @Override
    @TMITransactionalReadOnly
    public GEOUserImage getUserImage(int userId) {
        try {
            return usersRepository.getUserImage(userId);
        } catch (RuntimeException re) {
            throw re;
        }
    }
```

**EXEMPLO 5.1 – EXEMPLO DA UTILIZAÇÃO DE AUTO-WIRING EM TRANSAÇÕES.**

De notar que a variável "*usersRepository*" nunca é instanciada e está a ser usada em diferentes métodos de forma explicita. Isto porque foi injetado no "*container*" do Spring a classe "*usersRepository*" no arranque da aplicação no servidor, a mesma foi instanciada uma única vez e utiliza essa, e só essa, referência de instância em qualquer parte que este esteja a ser usado por "*auto-wiring"*. Uma única variável de memória pode ser utilizada em vários contextos. O uso do "*auto-wiring"* também traz a vantagem de não ter que ser adicionado ao ficheiro .xml de configuração do Spring toda a informação do "*usersRepository*", o que traria mais tempo de programação.

Toda a transação fica assegurada pelo Spring, inclusive o uso de exceções e controlo das mesmas. Quando há uma exceção na execução do código do servidor, e

que seja tratada para ser vista no cliente, esta irá ser processada pelo Spring sob a forma de "*RuntimeException*", e será esta a resposta ao invés de uma transação com sucesso. Esta exceção também é processada pela classe "*GWTServiceExporter*", uma classe que faz a gestão do serviço que foi pedido à base de dados, antes e depois, de o mesmo entrar nas camadas de BC.

## 5.4 Hibernate

O Hibernate é uma *framework* para o mapeamento objeto-relacional, em inglês, Object-Relacional Mapping (ORM), entre os atributos de uma base de dados tradicional e o modelo objeto Java de uma aplicação, a classe ou domínio.

O objetivo desta *framework* é diminuir a complexidade derivada da implementação de métodos que representem a estrutura da base de dados, bem como a leitura e escrita dos atributos dos mesmos nas tabelas.

O Hibernate recorre à reflexão, por anotações do Java, de modo a representar o mapeamento. Este mapeamento é transversal às bases de dados, ou seja, o código a implementar é independente da base de dados em execução.

Uma tabela que represente, por exemplo, utilizadores da aplicação será mapeada para um objeto Java correspondente a um utilizador da aplicação. Desta forma, todas as colunas da tabela e suas propriedades estão descritas no objeto, para que este seja utilizado na iteração com a base de dados.

Na forma tradicional, a inserção de um objeto na tabela de utilizadores teria o seguinte código Sql (5.2):

```
INSERT INTO GEO_USERS(ID, NAME, DESCRIPTION, MAL, PASSWORD, ACTIVE)
VALUES (1,'JOÃO','UTILIZADOR ADMINISTRATOR,'JOÃO@GMAIL.COM","($K$(D#K#',TRUE)
```

**EXEMPLO 5.2 – QUERY SQL QUE INSERE UM UTILIZADOR NA BASE DE DADOS.**

Este código Sql (5.2) seria depois passado como *String* no restante código Java que iria configurar uma transação para a atualização da base de dados, processar o Sql da *String* e retornar um resultado de sucesso ou falha.

Com recurso ao Hibernate apenas é necessário uma linha de código "*getCurrentSession().save(u)*" para inserir um objeto numa tabela (5.3). Todo o código extra, necessário para a criação de transações, desaparece, pois é o Hibernate que executa todos estes processos, baseado nas informações que foram configuradas no domínio que está a ser gravado.

```
@Override
public GEOUser insertUser(GEOUser u) {
    try {
        getCurrentSession().save(u);
    } catch (RuntimeException re) {
        throw re;
    }
    return u;
}
```

**EXEMPLO 5.3 – GRAVAÇÃO DE UM OBJETO DO TIPO UTILIZADOR NA BASE DE DADOS.**

Desta forma, torna-se relativamente fácil e acessível um único programador assumir o controlo de toda a gestão de uma base de dados, pois o código a criar será normalizado e transversal à mesma.

O objeto mapeado, no código "*GeoUser*" (5.3), é a classe que contem todos os atributos, que fazem corresponder às colunas da tabela, e suas propriedades. Este mapeamento é feito por anotações, como foi mostrado no exemplo (4.1).

A leitura de tabelas, incluindo *Joins* entre tabelas, deixa de ser complexo do ponto de vista programático, porque o Hibernate é que faz toda a gestão de cache da base de dados e mapeamento dos objetos.

Tomemos por exemplo uma ação que o utilizador faz na aplicação, supondo que o utilizador quer visualizar alguns pontos no gráfico de deslocamentos entre épocas de uma rede planimétrica. Depois de executado o pedido ao servidor para visualizar a informação é feita uma leitura à base de dados, no caso Sql sem recurso ao Hibernate, o utilizador teria que executar para cada objeto do domínio tantas *query's* SQL consoante as propriedades do mesmo. Com recurso ao Hibernate é utilizado o HQL (5.4), onde o programador utiliza apenas uma *query* recorrendo a linguagem de objetos na mesma, ficando ao cargo do Hibernate realizar em background todas as outras operações de conversão de Hql para Sql, fazer a *query* à base de dados e devolver os resultados finais para os passar para os objetos pedidos.

```
@Override
public GEOEpoch getEpoch(int epochId, boolean fetchOpticsObservations,
        boolean fetchOpticsPoints, boolean fetchOpoticsAdjustmentResults) {

    GEOEpoch ret = null;
    try {
        StringBuffer hqlQuery = new StringBuffer();

        hqlQuery.append(" from GEOEpoch epoch ");

        if (fetchOpticsObservations) {
            hqlQuery.append(" inner join fetch epoch.opticsObervationsPoints");
        }

        if (fetchOpticsPoints) {
            hqlQuery.append(" inner join fetch epoch.opticsPoints");
        }

        if (fetchOpoticsAdjustmentResults) {
            hqlQuery.append(" inner join fetch epoch.O2DResults");
        }

        hqlQuery.append(" where epoch.id = :epochId ");

        Query query = getCurrentSession().createQuery(hqlQuery.toString());
        query.setParameter("epochId", epochId);

        GEOEpoch ep = (GEOEpoch) query.uniqueResult();

        ret = ep;

    } catch (RuntimeException re) {
        throw re;
    }
    return ret;
}
```

**EXEMPLO 5.4 – EXEMPLO DE UMA QUERY HQL COM RETORNO DE UMA ÉPOCA E SUAS PROPRIEDADES.**

## 5.5 Sistema de Gestão de Bases de dados

O SGBD escolhido para o desenvolvimento desta aplicação foi o MySql Community Server®, por o mesmo ser gratuito e mantido pela Oracle Corporation®.

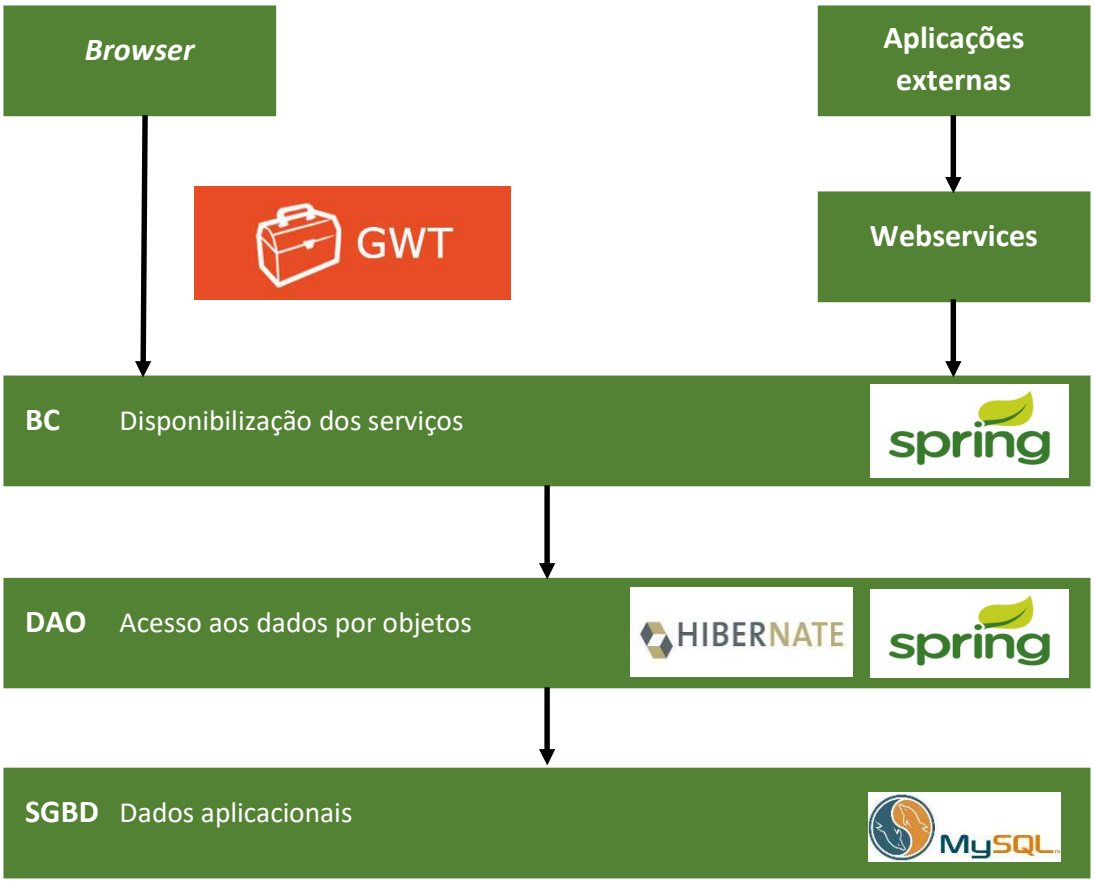
Apesar de existirem outros SGBD's como o PostgresSql® e o Oracle Server®, sendo este último não gratuito, a decisão caberia entre o MySql® e o PostgresSql®. O facto de ter já tido contacto com o MySql® em outras aplicações e referindo novamente que o mesmo é mantido pela Oracle, foi fator de desempate pelos dois.

Este SGBD possui várias ferramentas e conectores que permitem um uso alargado em outras aplicações. Os conectores são aplicações que comunicam entre as linguagens de programação e o SGBD, as ferramentas são aplicações que gerem o SGBD as suas tabelas e onde é permitido executar código Sql. É assim uma plataforma de gestão do SGBD.

O SGBD pode ser alterado em qualquer altura, sem impacto na aplicação, bastando nos ficheiros de configuração da mesma referir o SGBD que está a ser usado.

## 5.6 Tecnologias em camadas

A arquitetura da aplicação fica assim estabelecida com todas as ferramentas, tecnologias e *frameworks* que possam ser demostradas em camadas (5.10) e como as mesmas estão presentes na estrutura da aplicação.




ESQUEMA 5.10 – REPRESENTAÇÃO DAS TECNOLOGIAS DA APLICAÇÃO EM CAMADAS

# 6 Testes à Aplicação

Foram feitos testes à aplicação em todo o seu comportamento visual, bem como testes unitários (JUnit) ao código. JUnit é um framework que faz testes automatizados ao fluxo normal da aplicação. Os testes foram feitos seguindo a referência bibliográfica [1].

Para o nivelamento foram feitos os testes ao longo dos capítulos 14.4, 14.6. Os testes foram coincidentes com os do livro, no entanto variava o número de iterações do ajustamento dos parâmetros consoante a aproximação inicial os parâmetros.

Para as distâncias foram feito o teste ao longo dos capítulos 14.3, 14.4. Os testes foram coincidentes com os do livro, no entanto variava o número de iterações do ajustamento dos parâmetros consoante a aproximação inicial os parâmetros.

Para os azimutes foram feitos os testes ao longo dos capítulos 15.4, 15.5.1, 15.16 Os testes foram coincidentes com os do livro, no entanto variava o número de iterações do ajustamento dos parâmetros consoante a aproximação inicial os parâmetros.

Vão ser agora analisados ao pormenor os resultados da aplicação para alguns dos capítulos mencionados acima.

De modo a serem realizados os testes na aplicação, o utilizador deve fazer o registo no ecrã de entrada (6.1), de modo a fazer login de seguida.

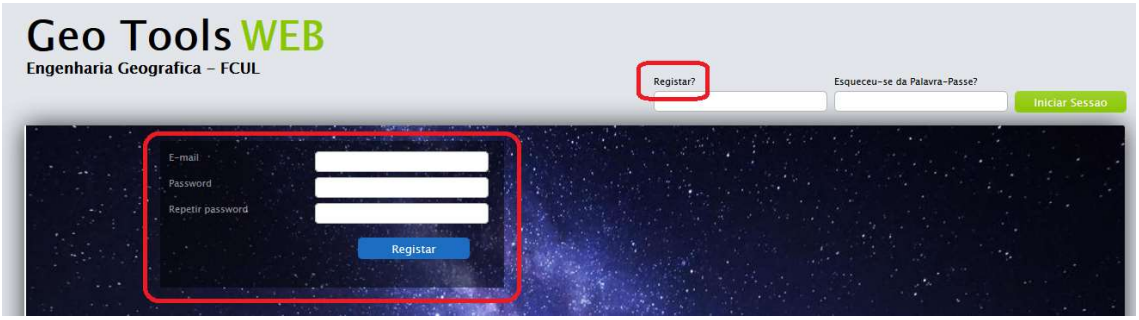



**FIGURA 6.1** – ECRÃ DE REGISTO E POSTERIOR LOGIN NA APLICAÇÃO.

Após o login o utilizador deve criar um novo projeto, que irá guardar todas as épocas de monitorização. Isto é feito carregando no botão da barra superior "Novo Projeto", que abrirá um ecrã para inserir os dados relativos ao projeto (6.2).

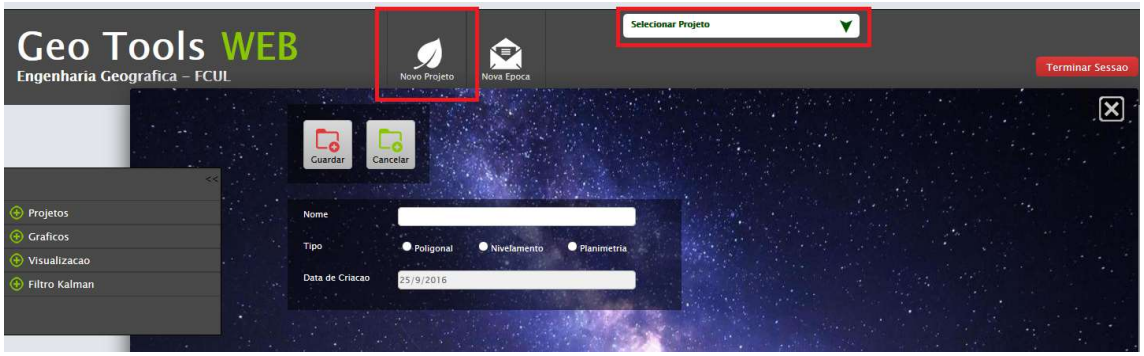



**FIGURA 6.2** – ECRÃ DE CRIAÇÃO DE UM NOVO PROJETO.

Durante a criação do projeto, o utilizador pode escolher entre planimetria, nivelamento geométrico e poligonal. Este será o tipo de projeto a utilizar durante a navegação na aplicação. No menu superior a branco o utilizador pode navegar entre os seus diferentes projetos.

Após a criação do projeto o utilizador deve começar por inserir uma nova época, no botão "Nova Época" do menu superior, ao lado do botão "Novo Projeto". Consoante o tipo de projeto é apresentado o respetivo ecrã de tratamento da informação.

No exemplo 14.5, uma rede que envolve apenas a observação de distâncias como mostra a figura:

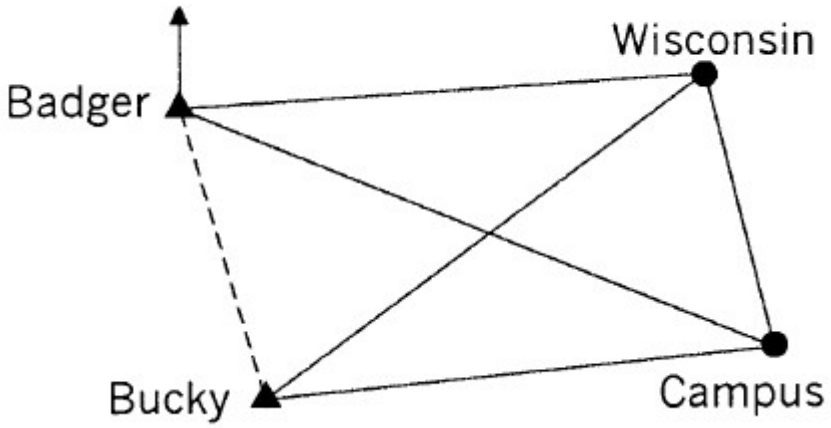



**FIGURA 6.3** – REDE OBSERVADA APENAS COM DISTÂNCIAS.

Quando o utilizador cria a nova época, baseado no projeto de planimetria selecionado, é apresentado o ecrã de tratamento da informação (6.4).

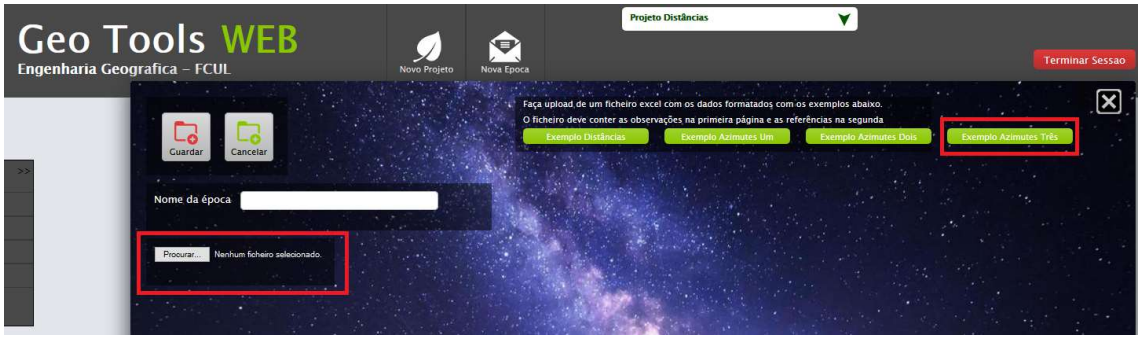



**FIGURA 6.4** – ECRÃ DE INSERÇÃO DE DADOS RELATIVOS AO PROJETO DE PLANIMETRIA.

Se o utilizador não tiver um ficheiro de entrada, pode fazer *download* de um dos quatro exemplos apresentados a verde. Este ficheiro é um ficheiro .xls (Excel), que deve ser preenchido com os dados a processar. Existem várias folhas no ficheiro, relativas a observações, referências e aproximações.

De seguida o utilizador deve fazer *upload* do ficheiro assinalado na caixa de "procurar ficheiro".

Quando for feito *upload* do ficheiro o mesmo será processado, se estiver incorretamente preenchido o utilizador recebe uma mensagem, caso contrário o programa continua o processamento dos dados e retorna os resultados (6.5).

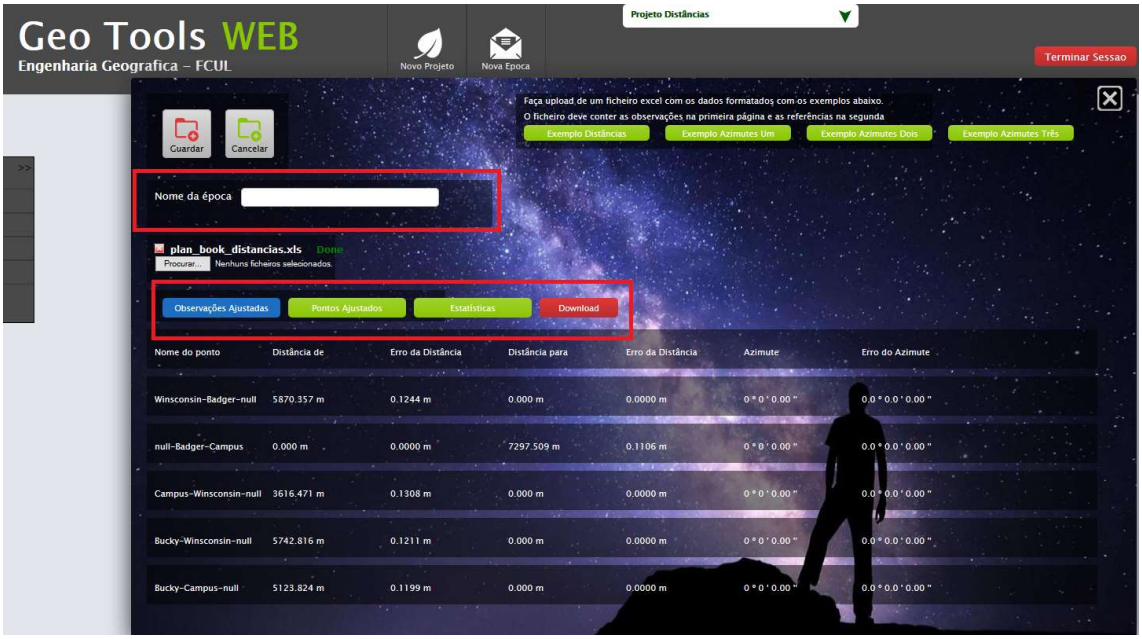



**FIGURA 6.5** – ECRÃ DE RESULTADOS DO PROCESSAMENTO DA INFORMAÇÃO.

Como pode ser visto na figura (6.5) o utilizador já tem os dados processados, significa que o ajustamento não deu erros, estando a ver as observações ajustadas, podendo navegar para os pontos ajustados, estatísticas e fazer *download* dos resultados, que incluem também as matrizes A e W das diversas iterações do ajustamento. Se o utilizador estiver satisfeito com os resultados, dá um nome à época e guarda a mesma. Ao adicionar novas épocas poderá comparar os gráficos de deslocamento nas opções do menu lateral, bem como seus projetos e épocas relacionadas.

Para o exemplo descrito as observações efetuadas (6.1) foram as seguintes:

| De | Estação | Para | Distância De | Precisão | Distância Para | Precisão |
|---|---|---|---|---|---|---|
| Winsconsin | Badger | | 5870,3020 | 1,0000 | | |
| | Badger | Campus | | | 7297,5880 | 1,0000 |
| Campus | Winsconsin | | 3616,4340 | 1,0000 | | |
| Bucky | Winsconsin | | 5742,878 | 1,0000 | | |
| Bucky | Campus | | 5123,7600 | 1,0000 | | |

**TABELA 6.1 – TABELA DE OBSERVAÇÕES.**

Os pontos de referência (6.2) têm as seguintes coordenadas M e P:

| | M | P |
|---|---|---|
| Badger | 2410000 | 390000 |
| Bucky | 2411820 | 386881,222 |

**TABELA 6.2 – TABELA DE REFERÊNCIAS.**

As aproximações para as coordenadas (6.3) dos pondos a determinar são as seguintes:

| | M | P |
|---|---|---|
| Winsconsin | 2415776,819 | 391043,461 |
| Campus | 2416898,227 | 387602,294 |

**TABELA 6.3 – TABELA DE APROXIMAÇÕES DE PONTOS A DETERMINAR.**

O processamento da rede pela aplicação obteve os seguintes resultados:

Número de iterações do ajustamento: **3**

Observações ajustadas (6.4):

| De | Estação | Para | Distância De | Precisão | Distância Para | Precisão |
|---|---|---|---|---|---|---|
| Winsconsin | Badger | | 5870,356684 | 0,1244 | | |
| | Badger | Campus | | | 7297,508989 | 0,1106 |
| Campus | Winsconsin | | 3616,470751 | 0,1308 | | |
| Bucky | Winsconsin | | 5742,816355 | 0,1211 | | |
| Bucky | Campus | | 5123,823927 | 0,1199 | | |

**TABELA 6.4 – TABELA DE OBSERVAÇÕES AJUSTADAS.**

Pontos Ajustados (6.5):

| | M | Precisão | P | Precisão |
|---|---|---|---|---|
| Winsconsin | 2415777 | 0,1488 | 391043,5 | 0,2206 |
| Campus | 2416898 | 0,1038 | 387602,3 | 0,2705 |

**TABELA 6.5 – TABELA DE PONTOS AJUSTADOS.**

Última iteração da matriz A e W, respetivamente:

| | | | |
|---|---|---|---|
| 0,984081 | 0,177723 | 0 | 0 |
| 0 | 0 | 0,944527 | -0,32843 |
| -0,30853 | 0,951214 | 0,30853 | -0,95121 |
| 0,689018 | 0,724744 | 0 | 0 |
| 0 | 0 | 0,990021 | 0,140917 |

| |
|---|
| 0,054684 |
| -0,07901 |
| 0,036751 |
| -0,06164 |
| 0,063927 |

No exemplo 15.6, uma rede que envolve apenas a observação de distâncias como mostra a figura (6.6):

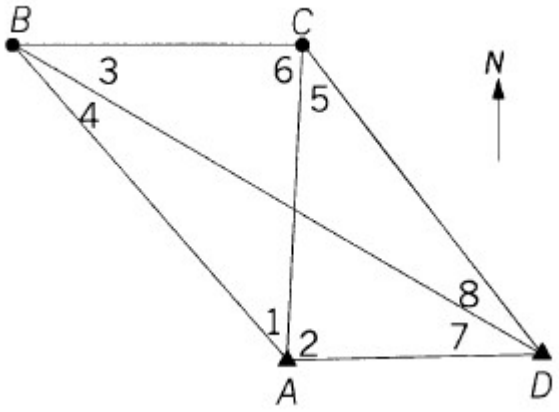



**FIGURA 6.6**– REDE OBSERVADA APENAS COM AZIMUTES.

O fluxo a seguir na aplicação é igual ao exemplo anterior, no entanto neste exemplo são ajustados ângulos.

As observações efetuadas (6.6) foram as seguintes:

| De | Estação | Para | Angulo Azimutal | Precisão |
|---|---|---|---|---|
| B | A | C | 42º35'29'' | 1 |
| C | A | D | 87º35'10,6'' | 1 |
| C | B | D | 79º54'42,1'' | 1 |
| D | B | A | 18º28'22,4'' | 1 |
| D | C | A | 21º29'23,9'' | 1 |
| A | C | B | 39º01'35,4'' | 1 |
| A | D | B | 31º20'45,8'' | 1 |
| B | D | C | 39º34'27,9'' | 1 |

**TABELA 6.6 – TABELA DE OBSERVAÇÕES.**

Os pontos de referência (6.7) têm as seguintes coordenadas M e P:

| | M | P |
|---|---|---|
| A | 9270,33 | 8448,9 |
| D | 15610,58 | 8568,75 |

**TABELA 6.7 – TABELA DE REFERÊNCIAS.**

As aproximações para as coordenadas dos pondos a determinar (6.8) são as seguintes:

| | M | P |
|---|---|---|
| B | 2403,6 | 16275,4 |
| C | 9649,8 | 24803,5 |

TABELA 6.8 – TABELA DE APROXIMAÇÕES DE PONTOS A DETERMINAR.

O processamento da rede pela aplicação obteve os seguintes resultados:

Número de iterações do ajustamento: **2**

Observações ajustadas (6.9):

| De | Estação | Para | Angulo Azimutal | Precisão |
|---|---|---|---|---|
| B | A | C | 42º35'31'' | 1 |
| C | A | D | 87º35'16'' | 1 |
| C | B | D | 79º54'37'' | 1 |
| D | B | A | 18º28'21'' | 1 |
| D | C | A | 21º29'26'' | 1 |
| A | C | B | 39º01'30'' | 1 |
| A | D | B | 31º20'52'' | 1 |
| B | D | C | 39º34'26'' | 1 |

TABELA 6.9 – TABELA DE OBSERVAÇÕES AJUSTADAS.

Pontos Ajustados (6.10):

| | M | Precisão | P | Precisão |
|---|---|---|---|---|
| B | 2403,5888 | 0,4689 | 2403,5888 | 0,4895 |
| C | 9649,8619 | 0,34474 | 9649,8618 | 0,8622 |

TABELA 6.10 – TABELA DE PONTOS AJUSTADOS.

Última iteração da matriz A e W (em segundos), respetivamente:

| | | | |
|---|---|---|---|
| -14,8914 | -13,0652 | 12,60517 | -0,29252 |
| 0 | 0 | -12,6052 | 0,29252 |
| 20,84421 | -0,28392 | -14,0457 | 11,93456 |
| 8,092912 | 1,414583 | 0 | 0 |
| 0 | 0 | 1,409352 | -4,40315 |
| -14,0457 | 11,93456 | 1,440527 | -11,642 |
| 6,798518 | 11,65064 | 0 | 0 |
| -6,79852 | -11,6506 | 11,19582 | 4,110625 |

| |
|---|
| 2,100989 |
| 5,032361 |
| -4,18338 |
| -1,41722 |
| 1,758123 |
| -5,40036 |
| 6,483821 |
| -1,47435 |

Sítio web: http://194.117.43.210:8080/GeoToolsWeb/

# 7 Conclusão

Apesar do pouco tempo que tive para dedicar a cada linha de código, a cada palavra de cada frase de um parágrafo desta tese valeu o esforço. O objetivo foi atingido e isso trouxe-me acima de tudo satisfação pessoal.

Este trabalho mostrou-me durante a sua realização que é possível automatizar vários processos associados à cadeia de ME, independentemente das técnicas, instrumentos e metodologias a usar.

Do ponto de vista tecnológico, este trabalho mostrou que o ritmo da evolução das tecnologias para soluções web está em constante crescimento, principalmente desde que a internet se tornou uma ferramenta de trabalho diária, que parece estar a tornar obsoletas ferramentas como o faxe ou até mesmo uma carta. Creio que é no acompanhar desta evolução que estará o futuro de muitas empresas da área da ME.

Com esta aplicação não mudei muito na realidade na ME, apenas e somente num servidor entre muitos da FCUL, no entanto essa mudança pode potenciar algo completamente diferente num futuro próximo, como em muitas outras áreas. Fica assim uma ferramenta moderna de apoio aos alunos do curso, onde os mais curiosos pela programação terão uma aplicação e uma explicação de como tudo foi implementado.

A aplicação ficará ao serviço da FCUL em que no futuro eu poderei eventualmente contribuir para algumas melhorias.

Dado que realizei a implementação desta aplicação de forma generalizada, em termos de código e metodologias associadas a esta, no futuro poderá ser acrescentado o processamento de informação de outros instrumentos ligados à área da monitorização. Como foi referida a utilização da metodologia MVP, isso permitirá no futuro não só uma aplicação de ME, mas eventualmente de outras áreas diferentes ainda que dentro da Engenharia Geográfica.

Alguns colegas e amigos já puderam visualizar e utilizar a aplicação, o que entre nós faz pensar que é possível elaborar um projeto muito mais ambicioso a médio e longo prazo, sendo esta primeira aplicação a base para aquilo que considero o futuro.

# 8 Referências Bibliográficas

**[1]** Gjilani, C., 2010, Adjustment Computations: Spatial Data Analysis, 5th Edition. John Wiley & Son, Ed., 672 pages, April 2010.

**[2]** Charles R. Farrar and Keith Worden, 2006, An Introduction to Structural Health Monitoring. 315 pages, December 2006.

**[3]** Christian Boller, 2009, Structural Health Monitoring – Na Introduction and definitions. Fu-Kuo Chang and Yozo Fujino, 2431 pages, 2009.

**[4]** José M. Torres, 2010, Métodos de observação durante a execução de obras subterrâneas. 204 pages, 2010.

**[5**] W.K. Chiu and S. C. Galea, 2010, Structural Health Monitoring: Research and Applications. 572 pages, December 2012.

**[6]** Stuart Turner, 1987, Lecture Notes in Earth Sciences. 12 - Applied Geodesy. 161 pages, 1987.

**[7]** J.L. Awange, E. W. Grafarend, B. Paláncz and P. Zaletnyik, 2010, Algebric Geodesy and Geoinformatics. 377 pages, 2010.

**[8]** V. B. Mendes, 1996 – Rev.1998a – Observações em Ciências Geográficas: Métodos de Ajustamento e Análise, 6. Avaliação Estatística de Resultados.

**[9]** Jeff Linwood and Dave Minter, 2010, Beginning Hibernate, an introduction to persistence using Hibernate 3.5, Second Edition. 379 pages, 2010.

**[10]** Hong Zhang, Y. Daniel Liang, 2006, Computer Graphics Using Java 2D and 3D. 632 pages, 6 December 2006.

**[11]** Bram Smeets, Uri Boness and Roald Bankras, 2008, Google Web Toolkit from novice to professional. 241 pages, 2008.

**[12]** Prabhakar Chaganti, 2007, Google Wen Toolkit, GWT Java Ajax Programing. 234 pages, 2007.

**[13]** Vipul Gupta, 2008, Accelerated GWT, Building Entreprise Google Web Toolkit applications. 291 pages, 2008.

**[14]** www.gwtproject.org

**[15]** https://spring.io/